# DRONE CYBERSECURITY MADE EASY

## FOR ABSOLUTE BEGINNERS

A Theoretical Guide to Firmware Analysis, Wireless Communication Protocols, Network Defense, Vulnerability Assessment, and Flight-System Protection



- Firmware Analysis
- Wireless Communication Protocols
- Network Defense
- Vulnerability Assessment
- Flight-System Protection





LAMAR C. STODOLA

# Drone Cybersecurity Made Easy for Absolute Beginners

A Theoretical Guide to Firmware Analysis, Wireless Communication Protocols, Network Defense, Vulnerability Assessment, and Flight-System Protection

**Lamar C. Stodola**

**Copyright © 2026 by Lamar C. Stodola**

All rights reserved.

No part of this book may be reproduced, distributed, or transmitted in any form or by any means without prior written permission from the copyright holder, except for brief quotations used in reviews or other uses permitted by applicable copyright law.

This book is intended for educational and informational purposes only. The author assumes no responsibility for damage, loss, or injury resulting from the use or misuse of the information provided. All cybersecurity testing should be conducted only on systems for which proper authorization has been obtained. Readers are responsible for complying with applicable laws, regulations, and safety requirements.

The information is provided in good faith and is not a substitute for professional engineering, cybersecurity, or aviation safety advice.

# Table of Contents

# The Modern Drone Security Landscape

## The Evolution of Unmanned Aircraft Systems and Their Digital Dependencies

Unmanned aircraft systems have developed from relatively simple remotely controlled machines into sophisticated platforms that combine aeronautical engineering, embedded computing, wireless communications, navigation technology, and automated decision-making. Their growing capabilities have expanded their use across agriculture, infrastructure inspection, environmental monitoring, emergency response, industrial logistics, scientific research, and public safety. As these applications have matured, the security requirements surrounding unmanned aircraft have become more complex. A modern drone is not simply an aircraft without a pilot onboard. It is a coordinated collection of hardware, software, communication links, data services, and human-operated systems whose interactions determine how safely and reliably the aircraft performs its assigned tasks.

Early remotely controlled aircraft relied heavily on direct operator input. The pilot issued commands through a radio transmitter, and onboard electronics translated those commands into motor movements or control-surface adjustments. Although these systems still depended on electrical components and communication signals, their operational logic was comparatively limited. The aircraft generally performed only the functions necessary to respond to the operator, maintain basic stability, and execute relatively simple maneuvers. Security considerations therefore centered primarily on

the integrity of the control link, the reliability of the onboard electronics, and the possibility of unauthorized interference.

The introduction of increasingly capable flight controllers changed this arrangement. Microprocessors became powerful enough to execute sophisticated control algorithms, interpret multiple sensor inputs, and maintain stable flight without continuous manual correction. Inertial measurement units supplied information about acceleration and angular velocity, while barometers, magnetometers, and satellite-navigation receivers provided additional data about altitude, orientation, and position. The flight controller combined these inputs to estimate the aircraft's state and calculate the adjustments needed to follow the pilot's commands or an autonomous mission.

This transition introduced an important distinction between controlling an aircraft and supervising an aircraft. In a manually controlled system, the operator may provide frequent corrections throughout a flight. In an autonomous system, the operator can instead define a mission, establish operating limits, and allow onboard software to manage many intermediate actions. The resulting reduction in direct human intervention improves operational efficiency, but it also increases dependence on software correctness, sensor reliability, configuration integrity, and carefully designed failure-handling procedures.

Autonomy does not mean that a drone acts independently of every external influence. Its decisions remain bounded by the capabilities of its sensors, the logic of its software, its configured operating parameters, and the information available to it. A navigation algorithm cannot reliably compensate for every possible sensor failure, and a flight controller cannot guarantee safe behavior if its inputs are corrupted or its assumptions are invalid. The aircraft's ability to respond appropriately to an abnormal condition depends on how that condition was anticipated during design and how

effectively the system recognizes that its normal operating assumptions no longer hold.

The expansion of digital connectivity has introduced another layer of dependence. Many aircraft now exchange telemetry with ground control applications, receive mission information from external computers, communicate through local wireless networks, and integrate with cloud-based management platforms. Some systems use cellular connectivity for remote supervision, while specialized applications may depend on satellite communications for operations beyond the practical reach of conventional radio links. Cameras and other payloads can transmit substantial volumes of information to operators or remote processing services. Maintenance applications may collect diagnostic records, and fleet-management systems may store aircraft identities, configuration information, flight histories, and mission assignments.

These connections create operational advantages, but they also extend the number of components that must be trusted. A defect in a ground control application can affect the reliability of the information presented to an operator. A compromised management account can expose sensitive mission records or permit unauthorized changes to configurations. An insecure software update process can introduce malicious or defective code into a previously reliable device. A poorly protected communication interface may allow an unauthorized party to access information or attempt actions that the system was not designed to expose.

Security must therefore be considered throughout the operational life of an unmanned aircraft. The relevant questions begin before a drone enters service. How was its firmware produced? Which components and libraries does it depend on? How are updates authenticated? What happens when the aircraft loses communication with its operator? Which credentials authorize mission changes? How are logs protected against unauthorized modification? What evidence exists that safety mechanisms behave

as intended? These questions remain important during deployment, maintenance, software upgrades, and eventual retirement.

The distinction between cybersecurity and operational safety is particularly significant. An ordinary software defect may cause an application to display incorrect information or terminate unexpectedly. In a drone, a similar defect can have consequences that extend into physical space. A control-system failure may affect stability, an incorrect navigation estimate may cause an aircraft to deviate from its intended route, and an unauthorized configuration change may invalidate an established safety limit. Conversely, not every aircraft malfunction is a cybersecurity incident. Mechanical wear, electromagnetic interference, poor maintenance, battery degradation, and environmental conditions can produce failures without any malicious activity.

Professional assessment must distinguish these causes rather than treating every abnormal behavior as evidence of an attack. Security analysis becomes more useful when it considers the complete engineering context, including ordinary failure modes, environmental limitations, human error, and deliberate attempts to exploit weaknesses. This approach also helps prevent unnecessary alarm when an observed fault has a more plausible explanation.

The digital development of unmanned aircraft has made this integrated perspective essential. Modern drone security is concerned not only with protecting individual devices but also with preserving the integrity of the relationships between aircraft, operators, software suppliers, communication providers, and supporting infrastructure. A secure aircraft operating within an insecure management environment may remain vulnerable, just as a well-protected network cannot compensate for an improperly implemented flight controller. Reliability depends on the coordinated protection of the entire system.

# Understanding the Architecture of Connected and Autonomous Drones

A connected drone is best understood as a system of interacting subsystems rather than a single electronic device. Its architecture typically includes the airframe, propulsion equipment, flight controller, navigation sensors, communication hardware, power system, payload, and one or more computing platforms. Outside the aircraft, a ground control station provides an interface for mission planning and supervision. Depending on the application, additional components may include remote servers, fleet-management applications, data-processing services, identity systems, and maintenance platforms.

The physical aircraft establishes the mechanical and electrical conditions under which every other component operates. Motors, propellers, batteries, wiring, and structural elements determine the aircraft's ability to generate lift, maintain balance, and respond to control commands. Electronic speed controllers regulate motor operation in many multirotor designs, receiving commands from the flight-control system and adjusting electrical power accordingly. Failures in these components can affect flight even when the software and communication systems are functioning correctly.

The flight controller is responsible for translating desired aircraft behavior into control outputs. It processes measurements from onboard sensors, estimates the aircraft's current state, and applies control algorithms to reduce the difference between the measured state and the desired state. These operations commonly occur at carefully managed intervals because delays or inconsistent timing can degrade control performance. The flight controller must also manage operating modes, interpret relevant commands, and respond to conditions such as low battery voltage, navigation uncertainty, or communication loss.

Many flight controllers use a real-time operating system or another environment designed to provide predictable execution of time-sensitive tasks. Real-time behavior does not necessarily mean that every operation must complete instantly. It means that important tasks must meet their timing requirements consistently enough to preserve correct system behavior. A process that produces the correct result too late can be just as problematic as one that produces an incorrect result. Consequently, security measures that consume excessive processing resources or introduce unpredictable delays must be evaluated carefully in flight-critical software.

Navigation systems provide information that supports position estimation, route following, and automated flight. Satellite-navigation receivers estimate position and velocity from signals transmitted by navigation satellites. Inertial sensors measure acceleration and angular motion, while other sensors can provide altitude, heading, optical flow, or visual information. The flight controller or a dedicated navigation computer combines these observations according to the system's design.

Every sensor has limitations. Satellite-navigation performance can deteriorate when signals are obstructed or affected by interference. Inertial measurements accumulate error over time. Barometers can be influenced by environmental pressure changes, and magnetometers may produce misleading readings near sources of magnetic disturbance. A capable navigation system therefore considers the quality and consistency of its inputs instead of assuming that every measurement is accurate.

The security implications arise when a system accepts sensor information without adequate checks or relies too heavily on a single source of truth. A reading can be technically valid in format while being unreliable in meaning. A navigation message may contain plausible coordinates even when the underlying position estimate is wrong. A sensor can remain electrically functional while producing

biased measurements. Good system design incorporates plausibility checks, redundancy where appropriate, and defined responses when confidence in an estimate falls below acceptable limits.

Communication architecture introduces another set of relationships. A drone may use separate channels for operator commands, telemetry, payload data, maintenance, and internet connectivity. In some systems, several of these functions share a physical radio or network interface. Others use multiple independent links. The architecture determines which components can communicate, which messages they are permitted to exchange, and what happens when a connection fails.

Telemetry generally includes information about the aircraft's status, such as position estimates, battery conditions, operating mode, sensor readings, and mission progress. Command traffic carries requests to change operating modes, adjust settings, or initiate authorized actions. Payload traffic may include images, video, scientific measurements, or other mission data. Although these traffic categories can share infrastructure, they do not necessarily require identical permissions or protection mechanisms.

A sound design separates functions according to their operational importance. A camera application should not automatically receive unrestricted authority over flight-critical functions simply because both applications run on the same onboard computer. A remote monitoring service may need access to aircraft health information without requiring permission to modify navigation parameters. A maintenance technician may need to inspect diagnostic records but should not necessarily be able to change every operational setting.

These distinctions are examples of least privilege: each component receives only the permissions required to perform its assigned role. Implementing this principle can limit the consequences of a compromised application or misused account. However, separation must be enforced by actual technical controls. Naming two services

differently does not isolate them if both operate with unrestricted privileges or share credentials that permit access to the same resources.

Ground control stations provide another important architectural layer. They allow operators to plan routes, configure aircraft, monitor telemetry, and supervise missions. Depending on the system, the ground station may run on a dedicated controller, a laptop, a tablet, or an integrated console. It can also connect to other applications that process mission data or manage several aircraft.

The ground station is not merely a display. It may contain configuration files, authentication credentials, mission records, communication settings, and software capable of issuing consequential commands. If the operator's workstation is compromised, the attacker may be able to access information or capabilities that the aircraft itself exposes only indirectly. The security of the ground station must therefore be evaluated alongside the aircraft's embedded systems.

Cloud-connected deployments extend the architecture further. A management service may authenticate users, register aircraft, store telemetry, distribute approved configurations, and coordinate fleet operations. Cloud services can improve visibility and simplify maintenance, but they also introduce dependencies on account security, access-control policies, software interfaces, service availability, and the protection of stored information.

The resulting architecture is distributed. No single component necessarily possesses a complete picture of the system, and authority may be divided among the aircraft, operator, organization, and service provider. This makes explicit documentation especially important. Engineers must understand which component makes each decision, which component supplies the relevant data, and which controls prevent one subsystem from exercising authority beyond its intended role.

# Identifying Critical Assets, Trust Boundaries, and Security Assumptions

A professional security assessment begins by identifying what must be protected and why. In drone systems, critical assets include more than the aircraft itself. They may include flight-control software, firmware signing keys, operator credentials, navigation settings, mission files, communication secrets, maintenance records, payload data, and the integrity of safety mechanisms. The importance of each asset depends on the aircraft's purpose and operating environment.

For an agricultural drone, sensitive assets may include crop imagery, treatment maps, operational schedules, and configuration parameters associated with application equipment. An infrastructure-inspection aircraft may carry images of facilities, maintenance findings, and detailed location records. An emergency-response drone may transmit information about an incident, the location of responders, or the condition of affected infrastructure. A research platform may contain experimental software, proprietary sensor data, and unpublished scientific results.

These differences matter because security requirements must reflect operational priorities. Confidentiality protects information from unauthorized disclosure. Integrity protects information and system behavior from unauthorized or unintended alteration. Availability ensures that authorized users and systems can access required functions when needed. Authenticity helps establish whether a message, device, or identity is genuine. Accountability provides evidence that allows actions to be associated with responsible users or processes.

In drone operations, these properties frequently overlap. Unauthorized access to video feeds may violate confidentiality, while modification of mission data threatens integrity. Loss of a communication service may affect availability and, depending on the aircraft's behavior, operational safety. Falsified device identity can undermine authentication and make it difficult to determine which aircraft or service produced a particular message.

The consequences of an incident must also be considered. A stolen maintenance report may create a privacy or commercial problem without immediately affecting flight. An unauthorized change to a failsafe configuration may create a more direct safety concern. A compromised signing key can have wider implications because it may undermine confidence in software updates distributed to multiple devices. The assessment should therefore consider both the value of an asset and the potential reach of a compromise.

Trust boundaries describe the points at which information, authority, or responsibility passes between components with different security assumptions. A radio connection between an aircraft and a ground station forms one such boundary. So does the interface between an onboard application and the flight controller, the connection between a ground station and a cloud service, and the process through which a manufacturer or integrator supplies firmware to an operator.

Each boundary raises questions about identity, authorization, data validation, and error handling. How does the receiving component know who sent a message? How does it determine whether the sender is permitted to request the action? Does it validate the message's structure and values? What happens when information is incomplete, duplicated, delayed, or inconsistent? Are failures logged, and can those records be trusted?

A system may authenticate a user correctly while still granting excessive privileges. Similarly, an encrypted connection can protect data in transit without proving that the receiving application handles the data safely. Encryption, authentication, authorization, and input validation address different problems. Treating them as interchangeable creates gaps in the security design.

Security assumptions deserve equal scrutiny. An engineering team might assume that a local communication interface is accessible only to trusted personnel, that a maintenance laptop is free from malware, or that a network supplied by a service provider is adequately protected. These assumptions may be reasonable under specific circumstances, but they should be documented and tested rather than treated as permanent truths.

A particularly important assumption concerns physical access. Some systems rely on the idea that an attacker cannot reach exposed connectors, storage devices, or maintenance interfaces. That assumption may be valid for a sealed installation in a controlled facility but much less reliable for aircraft deployed at remote sites or shared among several operators. Physical protection can reduce risk, but it should not replace appropriate authentication, secure boot processes, and protection of sensitive information.

Another common assumption concerns trusted internal networks. Once a device joins an organization's network, it may be granted broad access to other systems. This approach can allow a compromised device to affect services that have little operational reason to trust it. A more defensible design evaluates each connection according to the identity of the communicating component, the requested action, and the sensitivity of the resource.

Trust must also be considered across the supply chain. A drone manufacturer may integrate a flight controller supplied by another company, use open-source software, and depend on third-party

libraries or externally produced communication modules. The security of the finished product therefore depends partly on components that the aircraft operator did not develop and may not be able to inspect completely.

Supply-chain confidence is strengthened by documented component inventories, verified software releases, secure development practices, signed firmware, vulnerability-management processes, and clear maintenance procedures. These controls cannot eliminate every uncertainty, but they make it easier to determine what is installed, where it came from, and how its integrity is maintained.

Identifying critical assets and trust boundaries provides a foundation for making security decisions proportionate to actual risk. Rather than applying the same controls indiscriminately, engineers can concentrate on the interfaces and components whose compromise would produce the most serious consequences.

## Mapping the Attack Surface Across Aircraft, Ground Stations, and Cloud Services

An attack surface is the collection of interfaces, components, and processes through which a system may be influenced, accessed, or exposed to unintended behavior. For a drone, this surface includes physical connectors, wireless communication interfaces, firmware update mechanisms, onboard applications, operator workstations, remote management services, and the human procedures used to operate and maintain the equipment.

The aircraft's embedded software is an important part of this surface. Firmware processes sensor readings, manages communications, interprets commands, and coordinates hardware behavior. Defects in parsing routines, memory handling, configuration management, or privilege separation can create security weaknesses. However, the presence of a software component does not by itself establish that the component is vulnerable. A meaningful assessment must examine the actual implementation, its exposure, the privileges it holds, and the conditions required for a defect to have an effect.

Firmware update mechanisms deserve particular attention because they can alter the software responsible for many other functions. A secure update process should establish the authenticity and integrity of the update package, verify compatibility, and handle interrupted or unsuccessful installations safely. Signed firmware can help prevent unauthorized modifications, provided that signing keys are properly protected and verification is enforced by the receiving device.

The update process must also consider recovery. An aircraft that cannot recover from a failed installation may become unavailable for service, while an update that silently accepts an invalid package may undermine system integrity. Security and reliability are therefore closely related. Updates should be authenticated, traceable, and tested under controlled conditions before deployment across operational fleets.

Physical interfaces create another category of exposure. Development and diagnostic connectors can provide legitimate access for manufacturing, maintenance, and troubleshooting. Depending on the hardware design and configuration, these interfaces may reveal diagnostic information or permit access to internal components. Their existence is not inherently a flaw; the concern is whether access is appropriately restricted and whether sensitive operations require additional authorization.

Wireless interfaces introduce risks associated with communication availability, message authenticity, and access control. A radio link can be affected by environmental interference or congestion even when no attacker is present. Security testing must distinguish ordinary radio-performance problems from malicious activity and must avoid disrupting unrelated communications. Authorized laboratory tests, recorded traffic, simulators, and shielded environments allow researchers to examine relevant behaviors without placing operational aircraft or nearby systems at risk.

Communication protocols should be evaluated according to how they represent data, authenticate participants, enforce permissions, and handle unexpected input. A protocol may define message formats without guaranteeing that every implementation uses them safely. Two compatible devices can differ substantially in how they validate commands, manage state, or respond to malformed messages. Assessment should therefore include both the protocol specification and the behavior of the actual software.

Navigation systems form a related area of concern. The integrity of position estimates affects route following, geofencing, and some automated safety responses. Researchers should examine how the aircraft identifies unreliable measurements, handles disagreements between sensors, and transitions to an appropriate operating mode when navigation confidence deteriorates. Such work is best performed in simulation or other controlled environments because deliberate interference with real navigation signals can affect systems beyond the test aircraft.

The ground control station presents its own set of risks. It may store mission files, communication parameters, user credentials, and records of previous flights. It may also rely on libraries, operating-system services, and software updates that introduce dependencies outside the aircraft itself. An outdated application or poorly

protected workstation can weaken the security of an otherwise well-designed aircraft.

Operator privileges require particular attention. A single account that can register aircraft, change safety-critical parameters, download sensitive records, and administer an entire fleet creates a broad concentration of authority. Role-based access control can reduce this exposure by assigning permissions according to responsibilities. Administrative actions should be logged, and access should be revoked when personnel no longer require it.

The surrounding network can influence the risk as well. A ground station connected to an office network may share infrastructure with email, document storage, remote administration tools, and other applications. If the station is compromised, network segmentation and restricted access can limit the attacker's ability to move toward additional resources. Conversely, unrestricted connectivity may turn a local workstation incident into a broader organizational security problem.

Cloud services extend the attack surface beyond the physical operating site. Fleet-management platforms may expose web interfaces, application programming interfaces, authentication services, and data-storage systems. Their security depends on correct authorization, sound software development, careful management of credentials, and protection against unauthorized access to stored information.

An API can be securely encrypted while still permitting a user to access another user's records because authorization checks are incomplete. A cloud account can require strong authentication yet remain exposed to excessive permissions assigned after login. A management service may also be vulnerable to configuration mistakes that expose data or permit unintended administrative actions. Assessments should examine the actual permissions and

trust relationships rather than assuming that the use of a cloud provider automatically establishes security.

Data protection must extend across the full information lifecycle. Flight logs, video, photographs, location records, and diagnostic reports may reveal sensitive operational patterns. Their collection should have a defined purpose, and their retention should reflect legitimate operational, legal, and safety requirements. Access should be limited, storage should be protected, and unnecessary records should not remain indefinitely available.

Human procedures are another part of the attack surface. Shared accounts, undocumented maintenance changes, unverified software installations, and informal handling of credentials can defeat technical controls. Security depends partly on whether operators and technicians understand their responsibilities and follow procedures consistently. Clear documentation, practical training, controlled configuration changes, and reliable incident-reporting processes help reduce these risks.

A useful attack-surface map records each component, its interfaces, its communication partners, the data it handles, the privileges it possesses, and the consequences of failure or compromise. This map should distinguish interfaces that are externally reachable from those that require physical access or privileged credentials. It should also identify dependencies that may not be obvious from the aircraft's physical design, including software-update services, identity providers, and remote data-processing systems.

The purpose is not to assume that every interface will be attacked. It is to establish a defensible picture of how the system can be reached and influenced, allowing security effort to be directed toward the most consequential weaknesses.

# Establishing a Threat Model for Professional Drone Security Research

A threat model converts general security concerns into a structured assessment of what could go wrong, under which conditions, and with what consequences. It provides a common basis for engineers, operators, software developers, and security researchers to discuss risks without relying on vague claims that a system is either secure or insecure.

The first task is to define the system under examination. A threat model for a small aircraft controlled directly by an operator will differ from one designed for autonomous industrial inspection or remote fleet management. The assessment should document the aircraft's intended mission, operational environment, communication architecture, software components, external dependencies, and safety requirements. It should also establish which components are included in the assessment and which are outside its scope.

The next task is to identify the assets that matter most. These may include the ability to maintain safe flight, the integrity of navigation and control settings, the authenticity of firmware, the confidentiality of payload data, and the availability of mission-management services. Asset identification should consider both direct and indirect effects. For example, the loss of a cloud service may not immediately affect an aircraft already flying, but it may prevent operators from retrieving essential information or preparing subsequent missions.

Threat actors should be described in terms of their capabilities, access, resources, and likely objectives. A threat model might consider an opportunistic individual with limited technical knowledge, a malicious insider with legitimate access, a criminal

seeking sensitive data, or a highly capable actor targeting a particular operation. These categories are analytical tools rather than predictions about a specific incident. Their purpose is to help determine which security assumptions remain reasonable under different conditions.

The model should distinguish between an actor's intention and actual technical capability. An individual may wish to interfere with an aircraft but lack access to its control interfaces. Another person may possess legitimate maintenance privileges without intending harm, yet accidentally introduce an unsafe configuration. Security planning should account for both deliberate misuse and unintentional actions that can produce similar consequences.

Potential threat scenarios can then be developed around the assets and trust boundaries already identified. Examples include unauthorized access to stored mission data, acceptance of an unauthenticated configuration change, installation of an unverified software update, exposure of credentials on a ground workstation, or loss of reliable navigation information. Each scenario should state the assumptions required for the event to occur and the conditions that could prevent or limit its effects.

Risk assessment must consider more than technical possibility. A weakness that requires unusual access, specialized equipment, and a narrow set of conditions may have a different risk profile from a remotely accessible defect affecting a widely deployed service. Likewise, a flaw that exposes non-sensitive diagnostic information may have different consequences from one that undermines a safety-critical function.

A common approach evaluates risk through the relationship between likelihood and impact. Likelihood should reflect available evidence, required access, system exposure, existing controls, and the plausibility of the scenario. Impact should consider confidentiality, integrity, availability, operational disruption, financial loss, privacy,

regulatory obligations, and physical safety. Numerical scores can help prioritize findings, but they should not create an illusion of precision when the underlying evidence is uncertain.

Drone security requires particular care when evaluating physical consequences. A cybersecurity weakness may have implications for safe flight, but the relationship must be demonstrated rather than assumed. Researchers should establish how the affected component influences aircraft behavior, whether safety mechanisms constrain the outcome, and whether the scenario is reproducible under controlled conditions. Where the consequences cannot be established confidently, the report should state the uncertainty and explain what further evidence would be required.

Mitigations should be mapped to specific threats. Authentication can reduce the risk of unauthorized access, while authorization limits what an authenticated user or component may do. Secure boot and verified firmware updates help protect software integrity. Network segmentation can reduce the impact of a compromised workstation. Independent monitoring may identify abnormal system behavior, while carefully designed failsafes can reduce the consequences of certain communication or navigation failures.

No single control addresses every threat. Encryption does not prevent an authorized user from misusing their privileges, and strong access control does not correct a defective flight-control algorithm. Redundancy can improve resilience but may introduce additional components and dependencies that also require protection. A good threat model considers the strengths and limitations of each mitigation and examines whether the controls themselves can fail.

Professional drone security research also requires explicit authorization and clear rules of engagement. Before testing begins, the researcher should establish ownership or permission, identify the devices and environments included in the assessment, define

permitted techniques, and document activities that could affect flight or external infrastructure. Live aircraft should not be subjected to intrusive experiments merely because a vulnerability is suspected. Simulation, isolated development hardware, recorded data, and non-destructive testing often provide a safer and more reproducible basis for investigation.

The rules of engagement should address data handling, operational safety, emergency stop conditions, evidence preservation, and communication with the system owner. If a test unexpectedly affects a safety-critical function or reaches an environment outside the authorized scope, the researcher should stop the activity and follow the agreed response procedure. This discipline protects the aircraft, the organization, and the credibility of the research.

Evidence quality is central to the final assessment. A finding should identify the affected component, describe the observed behavior, explain the conditions under which it occurs, and distinguish verified results from assumptions. Relevant evidence may include configuration records, software versions, controlled test logs, source-code observations, and simulation results. Sensitive credentials, personal information, and operational details should be handled according to the agreed data-protection requirements.

The resulting report should explain both the technical weakness and its operational significance. It should identify the affected assets and trust boundaries, describe existing safeguards, state the limitations of the assessment, and recommend proportionate remediation. Where a finding depends on a particular configuration or software version, that dependency should be made explicit. A vulnerability in one implementation should not automatically be attributed to every drone using the same general protocol or hardware category.

Threat modeling is not a one-time exercise. Aircraft configurations change, firmware is updated, new services are introduced, and operational practices evolve. A previously acceptable assumption

may become invalid after a new communication feature is enabled or an external management service is connected. Organizations should therefore revisit their models after significant architectural changes, major software updates, newly discovered vulnerabilities, and meaningful changes in the aircraft's operating environment.

A disciplined threat model provides a practical basis for prioritizing research and improving security without losing sight of the aircraft's primary purpose. It connects technical weaknesses to operational consequences, identifies the controls that matter most, and establishes a clear method for validating whether corrective measures have reduced risk.

# Flight Control Systems and Autopilot Architecture

## The Engineering Principles Behind Multirotor and Fixed-Wing Flight

The behavior of an unmanned aircraft is governed by a continuous interaction between aerodynamic forces, mechanical structure, electrical power, sensor measurements, and computational control. Although multirotor and fixed-wing aircraft can perform some similar missions, they achieve controlled flight through fundamentally different physical mechanisms. Understanding these differences is essential for evaluating autopilot behavior, diagnosing system faults, and assessing the security of the software responsible for keeping an aircraft stable.

Every aircraft in flight is subject to forces that include gravity, aerodynamic lift, thrust, and drag. Gravity acts toward the center of the Earth, while the other forces depend on the aircraft's design, motion, and interaction with the surrounding air. The aircraft's movement results from the combined effect of these forces and the moments they generate around its center of mass. A flight controller does not eliminate these physical forces. It adjusts the available control inputs to produce the desired response within the aircraft's mechanical and aerodynamic limits.

A multirotor aircraft generates most of its lift directly through powered rotors. Each rotor accelerates air downward, producing an opposing force that supports the aircraft's weight. By changing the

rotational speed of individual rotors, the flight controller can modify the total thrust and create moments that change the aircraft's orientation. This arrangement allows a multirotor to hover, move in different horizontal directions, and change altitude without requiring forward airspeed to generate lift in the same manner as a conventional fixed-wing aircraft.

The apparent simplicity of this arrangement conceals a demanding control problem. A multirotor is dynamically coupled, meaning that an action intended to change one aspect of its motion can affect other aspects as well. Tilting the aircraft changes the direction of its total thrust vector. If the aircraft tilts forward, for example, part of the available thrust acts horizontally, accelerating the aircraft forward, while the remaining vertical component supports its weight. Maintaining altitude during a maneuver may therefore require an increase in total thrust, provided the motors have sufficient reserve capacity.

The familiar quadrotor illustrates this principle. Four rotors are positioned around the aircraft's center of mass, usually in a symmetrical arrangement. Opposing rotors typically rotate in opposite directions to help balance reaction torques. The controller changes the relative thrust produced by these rotors to regulate roll, pitch, and yaw. Roll describes rotation around the longitudinal axis, pitch describes rotation around the lateral axis, and yaw describes rotation around the vertical axis.

The precise relationship between rotor commands and aircraft movement depends on the physical configuration. Rotor placement, motor response, propeller characteristics, battery voltage, total mass, and aerodynamic disturbances all influence the result. An aircraft carrying an additional payload may respond differently from the same aircraft without that payload because its inertia, center of gravity, and required lift have changed. A robust control system must operate within a defined range of these changing conditions rather

than assume that the aircraft always behaves exactly as it did during initial calibration.

Fixed-wing aircraft rely on a different source of lift. Their wings generate aerodynamic lift as air flows around their surfaces, with the resulting force depending on factors such as airspeed, air density, wing geometry, and angle of attack. The angle of attack is the angle between a wing's reference line and the relative airflow. Increasing it can increase lift within an appropriate operating range, but exceeding the critical angle of attack causes the airflow to separate sufficiently that lift deteriorates, potentially resulting in a stall.

Because a conventional fixed-wing aircraft requires suitable airflow over its wings to sustain flight, it generally cannot hover in place without specialized equipment or unusual aerodynamic arrangements. Its control system must account for airspeed, bank angle, pitch attitude, and available control-surface authority. Ailerons primarily influence roll, elevators primarily influence pitch, and rudders primarily influence yaw, although the precise effects depend on the aircraft's design and flight condition.

The distinction between attitude and flight path is particularly important. Attitude describes the aircraft's orientation, while the flight path describes its movement through space. A fixed-wing aircraft can point its nose in one direction while moving along a different path because of aerodynamic forces, wind, and momentum. A controller that regulates attitude must therefore use suitable guidance logic when the objective is to follow a route rather than merely maintain an orientation.

Multirotor and fixed-wing systems also differ in their energy requirements. A multirotor must continuously produce sufficient thrust to support its weight during a hover, consuming substantial electrical power for that purpose. A fixed-wing aircraft can use forward motion to generate lift and may achieve longer endurance for a given energy capacity, although actual performance depends on

airframe design, propulsion efficiency, payload, weather, and operating conditions. Hybrid aircraft attempt to combine the hovering capability of rotor-based flight with the efficiency of wing-borne flight, but they introduce additional transitions and control requirements.

These physical differences have direct implications for autopilot architecture. A multirotor controller typically manages rapid changes in rotor thrust, while a fixed-wing controller must account for aerodynamic response, airspeed, and control-surface effectiveness. A hybrid aircraft must coordinate these behaviors across different operating regimes. Security and reliability assessments must therefore consider the actual flight dynamics of the aircraft under examination. A control strategy that is appropriate for a quadrotor cannot simply be transferred to a fixed-wing aircraft without accounting for its different physical characteristics.

Environmental conditions further complicate the problem. Wind creates disturbances that can change the aircraft's motion relative to the surrounding air. Turbulence produces variations in aerodynamic forces, while temperature and air density affect propulsion and aerodynamic performance. Vibration can degrade sensor measurements, and a reduction in battery voltage may limit the thrust available from electric propulsion systems. These influences make flight control an ongoing process of measurement, estimation, and correction.

The engineering objective is not to make the aircraft immune to every disturbance. It is to design a system that remains stable and predictable within its intended operating envelope, recognizes when conditions exceed that envelope, and responds appropriately when normal operation can no longer be maintained.

# Flight Controllers, Control Loops, and Real-Time Decision-Making

The flight controller serves as the computational center of the aircraft's control system. It receives measurements from sensors, evaluates the current state of the aircraft, compares that state with the desired behavior, and calculates commands for the actuators. These operations repeat throughout the flight, forming a feedback-control system.

Feedback is essential because the aircraft cannot be controlled reliably through commands alone. A motor command does not guarantee a particular aircraft attitude, and a desired position does not establish that the aircraft has reached that position. Disturbances, mechanical differences, sensor errors, and changing operating conditions cause the actual response to differ from the expected response. Feedback allows the controller to measure these differences and make corrections.

A basic control loop contains a desired value, a measured or estimated value, an error calculation, a control algorithm, and an actuator response. Suppose a multirotor is expected to maintain a level attitude. The controller compares the desired orientation with the estimated orientation and calculates the correction required to reduce the difference. It then adjusts the rotor commands, causing the aircraft to rotate toward the intended attitude. New sensor measurements reveal the result, and the process continues.

The control system usually contains several interconnected loops rather than a single controller. An inner loop may regulate angular rates, while an outer loop regulates attitude. Higher-level loops can regulate velocity and position. Each layer has a different responsibility and operates on information appropriate to that task.

Angular-rate control responds to how quickly the aircraft rotates. Attitude control regulates its orientation. Velocity control attempts to achieve a desired speed and direction, while position control generates commands intended to move the aircraft toward a target location. Guidance logic may convert a mission route into a sequence of desired positions, velocities, or attitudes that the lower-level controllers can follow.

This layered structure separates immediate stabilization from broader mission objectives. A position controller does not generally need to determine the electrical behavior of each motor directly. Instead, it produces a desired motion that lower-level controllers translate into suitable actuator commands. The separation makes complex systems easier to develop and test, but it also creates dependencies. If a higher-level component supplies an invalid target, the lower-level controller may faithfully attempt to achieve an inappropriate objective unless suitable limits and validation mechanisms intervene.

Control algorithms must balance responsiveness, stability, and resistance to noise. A controller that reacts too slowly may allow disturbances to grow before corrective action occurs. A controller that reacts too aggressively can produce oscillation, excessive actuator activity, or sensitivity to measurement noise. The appropriate balance depends on the aircraft's physical characteristics, sensor quality, actuator response, and intended operating conditions.

Proportional-integral-derivative control, commonly known as PID control, remains an important method in flight-control engineering. Its proportional component responds to the current error, its integral component accounts for accumulated error, and its derivative component responds to the rate of change of error. Properly tuned, these terms can provide effective control for many aircraft systems. However, PID is not automatically suitable for every

control problem, and its performance depends on the quality of the measurements, the chosen gains, actuator limits, and the system's dynamic behavior.

Integral action illustrates the importance of implementation details. When an actuator reaches its limit, the controller may continue accumulating error even though additional corrective output is unavailable. This condition, commonly called integrator windup, can cause a delayed or excessive response when the actuator becomes effective again. Anti-windup mechanisms help prevent this behavior by limiting or correcting the accumulated integral term under appropriate conditions.

Real-time execution is another essential requirement. Flight-control tasks operate within timing constraints determined by the aircraft's dynamics and the design of its control loops. A mathematically correct calculation can still produce an unsafe result if it arrives too late to influence the relevant physical event. Predictable scheduling, controlled interrupt handling, appropriate task priorities, and efficient sensor processing all contribute to reliable operation.

The flight controller must also manage shared computing resources. Sensor acquisition, communication processing, navigation estimation, logging, and mission management may execute on the same processor. If a noncritical task consumes excessive processing time, a time-sensitive control task may miss its deadline. Engineers therefore need to consider worst-case execution time, memory use, scheduling interference, and the behavior of tasks under heavy load rather than relying only on average performance.

Watchdog mechanisms can help identify software that has stopped making expected progress. A watchdog may reset a processor or trigger a defined recovery action if a task fails to meet its timing or servicing requirements. Such mechanisms are useful, but they must be designed carefully. An indiscriminate reset during flight may itself create a dangerous interruption, particularly if essential control state

cannot be recovered safely. Recovery behavior should reflect the aircraft's physical capabilities and the consequences of interrupting control.

Command limits and actuator constraints also belong within the control architecture. Motors have finite thrust capacity, servos have limited travel, and actuators cannot respond instantaneously. A controller should not assume that every requested movement can be achieved. It must account for saturation, rate limits, and the possibility that several control objectives compete for the same physical resources.

For example, a multirotor may need to maintain altitude while correcting a substantial roll disturbance. Both objectives require available thrust, and the motors may not have enough reserve capacity to satisfy them fully. The controller must allocate actuator authority according to the system's design priorities and operating constraints. Such decisions become especially important near the limits of the aircraft's performance.

Real-time reliability also depends on how commands enter the control system. Commands may originate from a human operator, an autonomous mission planner, a position controller, or an emergency-management function. The architecture must define which source has authority in each operating mode and how conflicting requests are resolved. A valid command from one component should not automatically override a more important safety restriction.

From a security perspective, the control loop is both a technical mechanism and a trust boundary. The controller depends on the integrity of sensor data, configuration parameters, desired setpoints, and actuator commands. A defect in any of these elements can influence the aircraft's behavior. Nevertheless, it is important to distinguish an exploitable security weakness from an ordinary engineering defect. The assessment should establish the affected

interface, the conditions under which it can be influenced, and the practical consequences of that influence.

## Sensor Fusion, State Estimation, and Flight Stabilization

A flight controller cannot observe every aspect of an aircraft's motion directly. It receives measurements from sensors, each of which provides only partial information and has its own limitations. State estimation combines these measurements to produce a coherent representation of the aircraft's condition, including quantities such as orientation, angular velocity, position, and velocity.

An inertial measurement unit commonly combines gyroscopes and accelerometers. Gyroscopes measure angular velocity, while accelerometers measure specific force, which includes the effects of aircraft acceleration and gravity according to the sensor's reference frame. These measurements are available at high rates and are useful for stabilization, but neither sensor type provides a complete, drift-free description of motion on its own.

Integrating gyroscope measurements over time can estimate changes in orientation. However, small measurement biases accumulate, causing the estimate to drift. Accelerometers provide information related to gravity when the aircraft's motion is sufficiently limited or when other acceleration effects can be accounted for. During aggressive movement, the measured acceleration cannot be interpreted as gravity alone. A state estimator must therefore account for both the useful information and the uncertainty associated with each measurement.

Magnetometers can provide information about magnetic heading, but nearby electrical currents, ferromagnetic materials, and other magnetic disturbances can distort their readings. Barometers estimate pressure and can support altitude estimation, although changes in weather, airflow, and sensor placement may influence the measurements. Satellite-navigation receivers provide position and velocity estimates, but their quality depends on signal availability, geometry, environmental conditions, and receiver performance.

No single sensor is ideal for every operating condition. Sensor fusion combines complementary information to reduce uncertainty and improve the continuity of the estimated state. The system may use inertial measurements for rapid updates, satellite navigation for longer-term position correction, and barometric or other measurements to improve altitude estimation. The precise combination depends on the aircraft and its navigation architecture.

A Kalman filter is one widely used approach to state estimation. It predicts the system's state using a mathematical model and then corrects that prediction using incoming measurements. Extended and unscented Kalman filters accommodate certain nonlinear relationships, while other estimation methods may be used when their assumptions better fit the application. The important principle is that the estimator does not simply average all available readings. It considers the predicted behavior of the system and the expected uncertainty of the measurements.

The quality of a state estimate depends on the accuracy of the model, the calibration of sensors, the timing of measurements, and the validity of assumptions about uncertainty. If a sensor reports data with an incorrect timestamp, the estimator may combine observations that do not correspond to the same physical moment. If a sensor bias is not modeled appropriately, the estimator may become systematically inaccurate. If uncertainty is underestimated,

the system may place excessive confidence in a misleading measurement.

Calibration is therefore more than a routine setup procedure. It establishes the relationship between raw sensor output and the physical quantities the system intends to measure. Misalignment, scale errors, temperature effects, and bias can all influence the result. Calibration must be appropriate for the sensor and its installation, and the system should have ways to identify conditions in which the calibration may no longer be reliable.

Sensor health monitoring complements estimation. A measurement can be checked for reasonable magnitude, expected timing, continuity, and consistency with other observations. A navigation estimate that suddenly becomes inconsistent with inertial motion may warrant reduced confidence or a transition to a different operating strategy. However, a discrepancy does not automatically prove that a sensor has been attacked. Mechanical vibration, temporary signal loss, environmental disturbance, and ordinary hardware failure can produce similar symptoms.

The estimator should therefore represent uncertainty rather than force every measurement into a single unquestioned answer. Innovation checks, residual monitoring, plausibility limits, and sensor-quality indicators can help determine whether a new observation agrees sufficiently with the predicted state. When the disagreement exceeds an appropriate threshold, the system may reject the measurement, reduce its influence, or enter a degraded operating mode.

These mechanisms have direct security relevance. A maliciously manipulated measurement may remain within the numerical range expected by the software while being physically misleading. Simple range checks may not detect it. Cross-checking independent measurements and considering the aircraft's expected dynamics can improve detection, although independence must be evaluated

carefully. Two sensors that share the same power source, communication path, environmental vulnerability, or underlying data source may fail together.

Flight stabilization relies on the state estimate produced by these processes. If the estimated orientation is wrong, an attitude controller may issue commands that worsen the actual orientation. If position estimates are unreliable, an autonomous navigation system may calculate an inappropriate route correction. The controller can only respond correctly to the information available to it, so estimation quality is inseparable from control quality.

This relationship becomes particularly important when the aircraft transitions between operating conditions. During a rapid maneuver, the estimator must account for dynamic acceleration and changing sensor orientation. During a hover, it may depend more heavily on certain measurements for position and altitude stability. During a navigation outage, the system may continue estimating attitude while its confidence in absolute position declines. These distinctions should be reflected in the aircraft's operating logic and operator displays.

A robust system also considers the timing and integrity of sensor data. Stale measurements, duplicated messages, unexpected resets, or communication interruptions can distort the state estimate if they are not detected. Sensor drivers should validate the structure and status of received data, while higher-level estimation functions should consider whether the measurements are current and consistent with the expected update rate.

Testing state estimation requires more than observing whether an aircraft appears stable during a normal flight. Engineers can use simulation, recorded sensor data, controlled fault injection, and hardware-in-the-loop testing to examine behavior under unusual conditions. Relevant scenarios include sensor dropout, bias changes, delayed measurements, excessive vibration, inconsistent navigation

inputs, and processor load. Such testing helps establish whether the estimator identifies uncertainty and whether the control system responds appropriately when the estimated state becomes less reliable.

## Autopilot Software Architecture and Flight Mode Management

Autopilot software coordinates the many functions required to operate an unmanned aircraft. Its architecture commonly includes hardware drivers, sensor-processing components, state estimation, control algorithms, navigation and guidance logic, mission execution, communication handling, logging, and safety-management functions. These components exchange information through defined interfaces, allowing the system to coordinate immediate stabilization with higher-level operational objectives.

Hardware abstraction separates much of the flight logic from the details of individual electronic components. A driver may provide a consistent interface for obtaining gyroscope measurements even when different hardware models use different registers or communication methods. Similarly, an actuator interface can translate control outputs into commands appropriate for the installed motor controllers or servos. This separation improves portability and simplifies maintenance, but it also makes the correctness of interface contracts important.

Each software component should have a clearly defined responsibility. Sensor drivers should report measurements and status information. State estimation should provide an appropriately qualified estimate of the aircraft's condition. Guidance functions should generate desired motion based on the mission and operating constraints. Control functions should calculate corrections, while

actuator-management components should enforce hardware limits and translate the commands into physical outputs.

Mission management introduces a higher level of decision-making. It determines which task the aircraft should perform, which waypoint or objective comes next, and how mission progress is affected by events such as arrival at a target location or loss of a required capability. Mission software should not bypass lower-level safety constraints merely because a route or action has been requested. The architecture must define how mission objectives are constrained by the aircraft's current condition and operating limits.

Flight modes provide a structured way to select different control behaviors. A manual or stabilized mode may allow the operator to command movement while the autopilot assists with attitude stability. An altitude-controlled mode may regulate vertical position while leaving horizontal movement under operator control. A position-hold mode may attempt to maintain a location using suitable navigation information. An autonomous mission mode may execute a predefined route under specified conditions.

The exact names and capabilities of these modes vary among systems, and similar labels do not guarantee identical behavior. A mode that maintains altitude in one implementation may depend on different sensors or control assumptions from a mode with the same name in another. Technical documentation should describe the actual requirements and limitations rather than rely solely on familiar labels.

Mode transitions require particular care. Moving from manual control to position hold may require valid navigation estimates and a suitable operating condition. Entering an autonomous mission may require a valid mission definition, acceptable battery status, and confirmation that the aircraft can meet the task's requirements. If those prerequisites are not satisfied, the transition should be rejected or handled according to a clearly defined degraded mode.

The transition itself must be managed safely. Abrupt changes in setpoints can produce undesirable motion, especially if the new mode assumes a different interpretation of the operator's inputs. Controllers may need to initialize internal state, limit command changes, or synchronize their outputs to prevent a sudden jump. Engineers should examine both successful transitions and interrupted transitions rather than assume that the change of mode is merely a software label.

Authority management is equally important. Several components may be capable of generating commands, but the system must establish which source has precedence under each condition. A pilot's input, an autonomous mission request, a position controller, and an emergency-management function can have different priorities. If command arbitration is ambiguous, two components may issue conflicting requests or a lower-priority request may override a safety-critical action.

A well-defined architecture makes these relationships explicit. It identifies the permitted command sources, validates their requests, and enforces the applicable operating constraints before commands reach the actuators. Critical restrictions should be enforced at appropriate technical boundaries rather than depending exclusively on a user interface to prevent invalid actions.

Configuration management is another significant concern. Parameters influence controller gains, navigation behavior, failsafe thresholds, sensor interpretation, and mission limits. Incorrect values can degrade performance even when the software is otherwise functioning as intended. Configuration changes should be validated, recorded, and restricted according to their operational importance. Where practical, the system should reject values outside documented limits and preserve a known-good configuration for recovery.

Software updates can alter control behavior as well. A new version may correct a defect, introduce a feature, change a parameter's interpretation, or modify the interaction between components. Compatibility testing should consider the flight controller, sensors, actuator interfaces, ground station, and supporting tools as a coordinated system. An update that works in isolation may still create problems when integrated with a different hardware revision or an older configuration.

Logging and diagnostics provide essential evidence about software behavior. Useful records may include mode transitions, rejected commands, sensor-quality changes, failsafe activation, configuration modifications, and relevant timing or health indicators. Logging should be designed so that it does not interfere with time-critical control tasks. Records should also be protected against unauthorized modification where their integrity is important for investigations, maintenance, or regulatory obligations.

From a security standpoint, modular software architecture provides opportunities for isolation and controlled access. Components should not receive broader privileges than they need, and interfaces should validate incoming information. However, modularity alone does not guarantee security. If every component shares unrestricted memory access or a common set of powerful credentials, a defect in one module may still affect the entire system.

## Security Implications of Failsafes, Control Dependencies, and Fault Handling

Failsafes are mechanisms intended to reduce risk when an aircraft can no longer continue its normal operation safely. They may respond to low battery capacity, loss of the operator's control link, invalid navigation information, excessive attitude deviation, or other conditions identified by the system design. Their purpose is not to guarantee a safe outcome in every possible circumstance, but to provide defined responses when particular operating assumptions are no longer satisfied.

A failsafe is only as reliable as the conditions that activate it, the information used to evaluate those conditions, and the action selected in response. If the system cannot recognize that a communication link has failed, the associated failsafe may never activate. If the navigation estimate is unreliable, a response that depends on returning to a known position may be inappropriate. If the aircraft lacks sufficient energy to complete the selected recovery action, a technically correct decision may still be physically impossible to execute.

Communication-loss handling illustrates these dependencies. Some aircraft may be configured to hold their current position temporarily, return toward a designated location, land when conditions permit, or follow another predefined procedure. The appropriate response depends on the aircraft's capabilities, the quality of navigation information, the available energy, the surrounding environment, and the mission's safety requirements. No single response is universally correct for every aircraft and operating scenario.

A return procedure, for example, may depend on a valid home location, adequate navigation accuracy, sufficient battery reserve, and a route that remains suitable for the aircraft's current conditions. If one of these assumptions is invalid, the response may not achieve its intended purpose. A robust design evaluates the prerequisites for the selected action and defines what should happen when those prerequisites are not met.

Low-energy protection presents similar challenges. Battery estimates are affected by factors such as temperature, battery age, discharge rate, payload, and propulsion demand. A fixed threshold may not provide sufficient warning under every operating condition. A reliable system should consider how battery-health estimates are produced, how uncertainty is represented, and how much reserve is needed for the intended recovery action.

Failsafe behavior also depends on the control architecture. A safety-management function may request a mode change, but the flight controller must still be able to execute the required action. If the actuator system has failed, the processor is overloaded, or the state estimate is unreliable, the requested response may not be achievable. Engineers should therefore analyze the full path from fault detection through decision-making to physical action.

Fault handling begins with detection and classification. The system must distinguish between conditions that permit continued operation, conditions that require reduced capability, and conditions that demand an immediate safety response. A temporary loss of one noncritical sensor may justify continued flight with reduced functionality. A fault affecting essential attitude information may require a more restrictive response. These decisions depend on the aircraft's design and should be supported by documented safety analysis.

Graceful degradation allows a system to preserve essential functions when a component becomes unavailable. An aircraft may be able to continue stabilizing its attitude even when its confidence in absolute position declines, although it may no longer be suitable for autonomous navigation. This distinction can prevent unnecessary loss of control while avoiding the unsafe assumption that every capability remains available.

Redundancy can improve resilience, but redundant components must be evaluated for common-cause failures. Two sensors of the same type may be affected by the same environmental disturbance. Two communication links may depend on a shared power supply. A primary controller and a backup controller may use the same defective software or receive the same invalid input. Redundancy is most effective when the system understands these dependencies and provides a meaningful degree of independence.

Security failures can complicate ordinary fault handling. Unauthorized configuration changes may disable or weaken a safety feature. Malformed input may cause a software component to stop responding. A compromised application may repeatedly request an operation that consumes resources needed by a time-critical task. An update failure may leave a device in an unexpected state. These scenarios should be evaluated alongside mechanical and environmental faults because the aircraft experiences their physical consequences regardless of whether the original cause was malicious.

The reverse is also true: a security incident can trigger safety mechanisms that were designed primarily for accidental failures. If communication becomes unreliable, a failsafe may activate without knowing whether the cause is environmental interference, equipment malfunction, or deliberate disruption. The response should be based on the aircraft's current condition and the reliability of the available information, not on an unsupported assumption about the cause.

A critical principle is that safety mechanisms must not depend on a single unverified source of information when the consequences of an incorrect decision are severe. Where appropriate, systems can cross-check independent measurements, validate the origin and freshness of commands, restrict safety-critical configuration changes, and monitor whether requested actions produce the expected response.

The implementation must still account for the possibility that several signals become unreliable at the same time.

Fault containment limits the spread of a failure from one component to another. An application responsible for processing payload imagery should not be able to consume unlimited processor time or interfere arbitrarily with flight-control tasks. A communication parser should handle malformed messages without corrupting unrelated memory. A maintenance interface should not permit unverified configuration changes to bypass safety constraints. These controls reduce the likelihood that a local defect becomes a system-wide failure.

Recovery behavior deserves the same attention as initial fault detection. After a temporary sensor outage or communication interruption, the system must determine whether normal operation can safely resume. Restoring a link does not necessarily mean that every command received during the interruption remains appropriate. Re-entering an autonomous mode may require renewed validation of the aircraft's state, mission status, and operating conditions. The recovery process should avoid unexpected transitions or the execution of stale instructions.

Security testing of failsafes should be systematic and controlled. Simulation can reproduce communication loss, sensor faults, processor delays, navigation uncertainty, and low-energy conditions without exposing a live aircraft to unnecessary risk. Hardware-in-the-loop testing can add real flight-controller hardware to the simulated environment, allowing researchers to evaluate actual software and timing behavior while retaining control over the scenario.

Each test should have a defined purpose and expected outcome. Engineers should record the initial conditions, injected fault, detection time, selected response, resulting system behavior, and any deviation from the documented requirements. Testing should

include cases in which the expected failsafe cannot complete its preferred action, since a system that works only under ideal recovery conditions has not demonstrated adequate resilience.

A security assessment should also examine who can modify failsafe settings, whether such changes are logged, and whether critical parameters are protected from unauthorized alteration. Authentication alone is insufficient if any authenticated user can disable important protections. Configuration authority should be limited to the roles that genuinely require it, and significant changes should be traceable to an authorized process.

Operational documentation is essential because safety behavior cannot be inferred reliably from a feature name alone. Operators need to understand what triggers each failsafe, which prerequisites it requires, how the aircraft behaves when those prerequisites are unavailable, and what actions they are expected to take. Maintenance personnel need corresponding procedures for validating configuration integrity and verifying that safety functions remain operational after repairs or updates.

The strongest flight-control architecture treats security, reliability, and safety as connected engineering responsibilities. It protects the integrity of the information used by the controller, limits the authority of individual components, detects abnormal conditions, and defines responses that reflect the aircraft's physical capabilities. Its quality is demonstrated not merely by stable flight under normal conditions, but by predictable behavior when assumptions fail, components disagree, communication becomes unreliable, or the system must operate with reduced capability.

# Embedded Hardware and Drone Computing Platforms

## Microcontrollers, Processors, and Embedded Computing Architectures

The reliability, responsiveness, and security of an unmanned aircraft depend heavily on the computing hardware that coordinates its electronic and mechanical systems. Beneath the aircraft's visible structure lies an embedded computing environment responsible for processing sensor measurements, calculating control outputs, managing communications, supervising power conditions, and coordinating autonomous operations. Understanding this environment requires more than identifying the processor installed on a circuit board. It requires examining how computing resources are organized, how information moves between components, how timing requirements are maintained, and how hardware limitations influence the aircraft's overall behavior.

Embedded computing differs from conventional desktop computing because its software operates in close coordination with physical processes. A desktop application may tolerate a brief delay without affecting the physical environment, but a flight-control task must respond within predictable time constraints. The processor must obtain measurements, update the aircraft's estimated state, calculate corrective actions, and deliver actuator commands quickly enough to preserve the required control performance. These operations must continue even when communication traffic increases, diagnostic functions are active, or other software components encounter errors.

The microcontroller is a common foundation for flight-control hardware. A microcontroller typically integrates a processor core, memory, timers, communication peripherals, interrupt controllers, and other supporting functions within a single integrated circuit. This arrangement reduces the need for separate components and allows the designer to build a compact control system with relatively predictable behavior. Microcontrollers are particularly useful where deterministic timing, low power consumption, compact physical dimensions, and direct access to electronic interfaces are important.

A microcontroller's processor core executes instructions, while its memory stores program code and working data. Flash memory commonly retains firmware when power is removed, and static random-access memory provides temporary storage for variables, stack data, communication buffers, and other runtime information. Some devices also contain read-only memory for boot code or other fixed functions. The amount and organization of these resources influence which software architectures can be supported and how much room remains for additional functionality.

Memory limitations are important in flight-control systems. A control application must reserve enough space for its essential calculations, sensor data, task stacks, communication buffers, and fault-handling mechanisms. Excessive memory consumption can cause allocation failures, stack exhaustion, or instability. A design that functions correctly during routine testing may behave differently when logging volume increases or an unusual input triggers a more demanding execution path. For this reason, embedded software should be evaluated under representative worst-case conditions rather than only during ordinary operation.

Processor performance must also be understood in context. Clock frequency provides a useful indication of processing speed, but it does not fully describe a processor's ability to execute a particular workload. Instruction-set efficiency, memory latency, peripheral

design, interrupt overhead, compiler behavior, and the structure of the software all affect actual performance. A processor with a higher clock frequency may not outperform another processor for every task, particularly when the workload depends heavily on memory access or specialized instructions.

Flight controllers commonly use microcontrollers designed for real-time embedded applications. These devices may provide floating-point arithmetic, hardware timers, direct-memory-access controllers, watchdogs, and multiple communication interfaces. Floating-point hardware can accelerate calculations involving angles, sensor estimates, and control algorithms. Timers support precise scheduling and measurement, while direct memory access can move data between peripherals and memory without requiring the processor to handle every individual transfer.

These features help reduce processing overhead, but they must be configured correctly. An incorrectly managed direct-memory-access operation can expose stale or inconsistent data to a software task. An interrupt that occurs too frequently can consume processing time needed by more important work. A watchdog that is serviced without confirming that essential tasks are healthy may fail to detect a partial software failure. Hardware capabilities improve reliability only when the software uses them according to clearly defined requirements.

More computationally demanding aircraft may use a separate application processor or companion computer. Such a device can support a full operating system, sophisticated image processing, machine-learning workloads, mapping, computer vision, or advanced mission planning. Its greater memory capacity and processing resources allow it to perform tasks that would be impractical on a small flight-control microcontroller.

A companion computer should not automatically be treated as a replacement for the flight controller. The two devices often serve different purposes. The flight controller is responsible for time-

sensitive stabilization and actuator control, while the companion computer handles higher-level computation. They exchange information through a defined interface, such as a serial connection or network link. This separation can improve modularity, but it also creates a boundary where commands, configuration information, and computed results must be validated.

For example, a companion computer may calculate a desired waypoint or provide an object-detection result to a navigation application. The flight controller must still determine whether the resulting request is valid for the aircraft's current state and operating limits. A sophisticated external algorithm should not be granted unrestricted authority simply because its output appears technically advanced. The aircraft must retain appropriate safeguards against invalid commands, stale information, and loss of communication with the companion computer.

Computing architectures also differ in their ability to isolate tasks. A small microcontroller may run a bare-metal application, in which the main program and interrupt handlers coordinate all activity without a conventional operating system. Another design may use a real-time operating system that schedules separate tasks according to their priorities and timing requirements. Companion computers frequently run general-purpose operating systems that support multiple processes, filesystems, network services, and application frameworks.

Each approach presents different engineering trade-offs. Bare-metal software can be compact and predictable, but complex applications may become difficult to maintain as the number of functions grows. A real-time operating system can provide structured task management, synchronization, and resource allocation, although its configuration and use require care. A general-purpose operating system offers extensive functionality but introduces more software dependencies, services, and potential sources of resource contention.

The security implications follow directly from these architectural differences. A microcontroller with a small, carefully controlled software image may expose fewer interfaces than a companion computer running numerous services. However, limited functionality does not make the microcontroller inherently secure. Weak boot protections, exposed debug access, unsafe memory handling, or inadequate firmware validation can undermine an otherwise simple design. Likewise, a larger computing platform can be secured effectively when its services are minimized, permissions are restricted, and software is maintained properly.

Hardware security features may include secure boot support, memory protection, cryptographic acceleration, protected key storage, and mechanisms for restricting debug access. Secure boot helps establish that the software being executed has passed an integrity or authenticity check. Memory protection can prevent one task from accessing resources assigned to another, where the processor architecture and software environment support it. Cryptographic acceleration can improve the efficiency of authentication and encryption operations, while protected storage can reduce the exposure of sensitive credentials.

These features have limitations. Secure boot cannot guarantee that correctly signed software is free from defects. Cryptographic acceleration does not protect keys that are stored insecurely elsewhere. Memory protection cannot compensate for every logic error or prevent all forms of denial of service. Effective security requires a combination of hardware capabilities, appropriate software design, sound configuration, and a maintenance process that addresses newly discovered weaknesses.

Power consumption also influences processor selection. The computing system draws energy from the aircraft's electrical supply, and unnecessary consumption reduces the energy available for flight. More powerful processors can support advanced functions but may

require additional power regulation, thermal management, and electromagnetic design work. The resulting hardware must operate reliably despite vibration, changing temperatures, electrical noise, and limited cooling.

The choice of computing platform should therefore reflect the aircraft's actual requirements. A basic aircraft requiring reliable stabilization may need a different architecture from a long-range inspection platform carrying cameras and advanced onboard processing. Engineers must balance computational capacity, timing predictability, energy consumption, maintainability, physical size, and security requirements rather than selecting hardware based on a single performance specification.

## Flight Controller Boards, Peripheral Interfaces, and System Buses

A flight controller board provides the physical platform on which the processor, power regulation, sensors, communication interfaces, and supporting electronics operate together. Its layout influences signal quality, thermal behavior, electromagnetic compatibility, maintenance access, and the reliability of connections between components. Two boards using similar processors may perform differently because of differences in circuit design, component selection, firmware support, and integration quality.

The processor is only one element of the board. Voltage regulators provide the electrical levels required by the processor and peripherals. Oscillators or other clock sources establish timing references. Connectors link the board to sensors, actuators, communication equipment, and power systems. Filtering and protective components help manage electrical disturbances. The

arrangement of these elements affects the board's ability to operate reliably in the electrically noisy environment of an aircraft.

Peripheral interfaces allow the processor to exchange information with external devices. Common interfaces include serial communication, inter-integrated circuit communication, serial peripheral interface, universal asynchronous receiver-transmitter connections, and general-purpose input/output pins. Their characteristics differ in speed, wiring requirements, addressing, synchronization, and electrical behavior. Choosing an interface requires consideration of the peripheral's needs and the constraints of the physical installation.

A universal asynchronous receiver-transmitter, commonly abbreviated as UART, supports serial communication between devices using separate transmit and receive signals, often with a common reference ground. It is widely used for telemetry modules, diagnostic consoles, and other embedded peripherals. Because UART itself does not necessarily provide message authentication or encryption, the security of a connection depends on the protocols and controls implemented above the electrical interface.

The inter-integrated circuit bus, commonly called I²C, allows multiple compatible devices to share data and clock lines. It is often used for onboard sensors and other relatively short-distance communications within electronic assemblies. Device addressing and shared wiring make the interface convenient, but bus behavior can be affected by electrical loading, wiring length, pull-up resistance, and the characteristics of connected devices. A malfunctioning peripheral may interfere with communication for other devices sharing the same bus.

The serial peripheral interface, or SPI, commonly uses separate clock, data, and device-selection signals. It can support relatively high communication rates and is frequently used with sensors, memory devices, and other embedded components. Its electrical

connections and transaction structure differ from I²C, and the exact implementation depends on the devices involved. Engineers must verify that timing, voltage levels, and transaction requirements are compatible rather than assuming that devices using the same general interface will communicate correctly without configuration.

General-purpose input/output pins provide flexible digital signals that software can read or control. They may be used to detect discrete states, activate peripheral functions, or support board-specific signaling. Some pins have alternate functions tied to timers, communication peripherals, or interrupt inputs. Incorrect configuration can cause conflicts between functions, and an exposed pin may provide access to internal behavior that was intended only for manufacturing or maintenance.

System buses connect the processor to memory, peripherals, and other internal resources. Their design determines how data moves, how competing transactions are handled, and which components can access particular resources. On complex processors, the interconnect may support several masters, including the processor core and direct-memory-access controllers. This improves throughput but increases the importance of access control and correct coordination between concurrent operations.

Timing is a recurring concern in bus-based systems. A sensor may produce new data at a defined rate, but the processor must still retrieve it, interpret it, and deliver the result to the relevant control task. Bus contention, repeated retries, electrical faults, or poorly designed drivers can introduce delays. Where the affected data supports flight stabilization, these delays may influence control performance.

Board layout can also influence measurement quality. Switching regulators, high-current motor connections, digital clock signals, and radio transmitters can generate electrical noise. Sensitive analog circuits may be affected by poor grounding, inadequate filtering, or

unsuitable component placement. Inertial sensors can be influenced by mechanical vibration transmitted through the mounting structure. These factors can create misleading readings even when the software is functioning as intended.

A professionally designed board accounts for electrical and mechanical conditions as well as computational requirements. Signal paths, grounding, connector retention, power distribution, and component placement must be suitable for the intended environment. Flight controllers mounted on aircraft experience vibration and temperature changes that are uncommon in ordinary office electronics. Secure connectors and suitable strain relief can prevent intermittent faults that might otherwise appear to be software or communication problems.

Debugging and maintenance interfaces require deliberate treatment. Development boards commonly provide access to debugging facilities because engineers need to inspect memory, set breakpoints, and diagnose failures. Production equipment may retain some of these interfaces for legitimate servicing. The security concern arises when access remains unrestricted despite the device being deployed in an environment where unauthorized physical access is plausible.

A debug interface can expose sensitive information or provide substantial control over the processor, depending on the device and its configuration. Manufacturers may restrict debug access through hardware settings, authentication mechanisms, lifecycle controls, or other protections. Such restrictions should be evaluated against legitimate maintenance requirements so that security controls do not prevent necessary recovery or inspection.

Peripheral configuration should also be documented. An interface assigned to one sensor may be reassigned during a hardware revision, and a pin used for diagnostics on one board may have a different role on another. Incorrect assumptions can lead to failed communication, unintended electrical behavior, or unexpected

software access. Accurate board documentation and version tracking are therefore important for both maintenance and security assessment.

The relationship between hardware and firmware is especially important when evaluating vulnerabilities. A software component may assume that a peripheral always returns a response within a particular period or that a connected device follows a documented message format. If those assumptions are not enforced, unexpected peripheral behavior can cause errors elsewhere in the system. Conversely, a hardware protection feature may be ineffective if the firmware configures it incorrectly or leaves it disabled.

A complete assessment should therefore consider the board as an integrated system. Processor specifications, interface assignments, power circuitry, sensor placement, debug access, and firmware configuration all contribute to the actual security and reliability of the device.

## Inertial Sensors, Barometers, Magnetometers, and Navigation Receivers

Sensors provide the measurements that allow a flight controller to estimate the aircraft's motion and position. Their output influences stabilization, altitude control, navigation, and automated mission execution. A sensor's usefulness depends not only on its nominal accuracy but also on its calibration, installation, sampling rate, noise characteristics, environmental sensitivity, and interaction with other components.

An inertial measurement unit generally combines gyroscopes and accelerometers. A gyroscope measures angular velocity around one or more axes, while an accelerometer measures specific force. These measurements support the estimation of attitude and motion, particularly when rapid updates are required. Their small size and high sampling rates make them suitable for embedded flight-control applications.

Gyroscope bias is a significant source of estimation error. Even a small bias can accumulate when angular rates are integrated over time. Temperature changes may affect the bias, and vibration can introduce additional noise. The system may estimate and compensate for bias, but the quality of that compensation depends on the sensor, calibration process, estimator design, and conditions under which the aircraft operates.

Accelerometers provide useful information about gravity when the aircraft's acceleration is appropriately accounted for. During dynamic movement, however, the sensor responds to combined effects that cannot be separated by simply treating every measurement as a gravity vector. The estimator must combine accelerometer information with other observations and a model of the aircraft's motion to produce a reliable orientation estimate.

Barometers estimate atmospheric pressure and can be used to derive relative altitude changes. They are valuable because they provide information independent of inertial integration, but they are not direct measurements of geometric altitude. Weather-related pressure changes, airflow around the aircraft, enclosure design, and temperature can influence their readings. A flight controller must interpret barometric data in context and avoid assuming that every pressure change corresponds exactly to a change in height.

Magnetometers measure magnetic-field direction and strength. When properly calibrated and installed, they can provide heading

information relative to the local magnetic field. Nearby electrical currents, motors, magnetic materials, and poorly arranged power wiring can distort that field. Calibration performed in one environment may not compensate fully for disturbances introduced by a different payload or hardware configuration.

Navigation receivers commonly use signals from global navigation satellite systems to estimate position and velocity. Their performance depends on signal availability, satellite geometry, receiver quality, antenna placement, and the surrounding environment. Buildings, terrain, structures, and other obstructions can reduce signal quality. Reflections may introduce multipath errors, while interference can degrade or interrupt reception.

A navigation receiver does not independently guarantee that the reported position is trustworthy. The receiver produces an estimate based on the signals it can process and the assumptions built into its algorithms. A system that relies on navigation data should consider quality indicators, continuity, consistency with other sensors, and the operational consequences of uncertainty.

Sensor placement is as important as sensor selection. An inertial sensor mounted near a source of vibration may produce less useful measurements than a similar device installed with appropriate mechanical isolation. A magnetometer placed near high-current wiring may experience persistent magnetic interference. A navigation antenna with poor visibility of the sky may produce unreliable position estimates even when the receiver itself is functioning correctly.

The design must also account for coordinate frames. Sensor axes, aircraft body axes, and navigation reference frames may not align naturally. Software uses calibration parameters and coordinate transformations to interpret the measurements correctly. Errors in these transformations can cause systematic inaccuracies that are

difficult to diagnose because the raw sensor readings may appear reasonable.

Sensor fusion combines measurements to produce a more complete estimate of the aircraft's state. Inertial measurements provide rapid updates, while navigation receivers and other sensors provide complementary information. Estimation algorithms account for uncertainty and use predicted system behavior to evaluate incoming observations. This process can improve continuity and accuracy, but it cannot eliminate every source of error.

The security implications of sensor processing are substantial. A sensor message may be syntactically valid yet inconsistent with the aircraft's actual state. A measurement may be stale, repeated, or affected by a fault. Software should validate data format, timing, range, and consistency where appropriate. It should also retain enough diagnostic information to help engineers distinguish sensor faults from communication problems or estimator defects.

A useful assessment examines how the system behaves when one sensor becomes unavailable or unreliable. Does the aircraft continue with reduced capability? Does the estimator increase its uncertainty? Is the operator informed of the degraded condition? Does the system prevent autonomous actions that require reliable position information? These questions are more informative than checking whether a sensor produces plausible values during ordinary operation.

Sensor redundancy can improve resilience, but only when the redundant sources provide meaningful independent information. Two identical sensors mounted next to one another may share the same environmental disturbance. Two navigation receivers using the same antenna or exposed to the same interference may fail together. The system should consider these common dependencies when determining how much confidence to place in redundant measurements.

Maintenance also affects sensor reliability. Mechanical impacts, changes in payload mounting, replacement of a flight controller, or alterations to power wiring can change the behavior of sensors. Calibration and validation should therefore be repeated when required by the manufacturer's procedures or by a material change in the aircraft's configuration. Maintenance records should identify the hardware revision, calibration status, and relevant software version.

## Electronic Speed Controllers, Motor Control, and Power Distribution

The propulsion system converts electrical energy into the mechanical force required for flight. In many electrically powered drones, electronic speed controllers regulate the power delivered to brushless motors. They receive commands from the flight-control system and adjust motor operation according to the supported control interface and the controller's internal design.

A brushless motor generates torque through interactions between its magnetic fields. The electronic speed controller switches current through the motor's windings in a sequence that produces rotation. The precise method of commutation depends on the motor and controller design. In many systems, the controller estimates rotor position or uses other feedback and timing methods to coordinate the electrical switching.

The flight controller generally does not regulate each motor by directly switching its power transistors. Instead, it calculates the required actuator outputs, and the electronic speed controllers translate the corresponding commands into electrical behavior. This division of responsibility allows the flight controller to focus on

aircraft-level control while the motor controllers manage lower-level electrical switching.

The quality of this interface matters. Command formats, update rates, signal levels, error behavior, and supported feedback information vary among systems. Some controllers provide telemetry about rotational speed, current, temperature, or fault conditions, while others provide more limited feedback. The flight-control software should interpret available feedback correctly and should not assume that a requested motor output has been achieved simply because a command was transmitted.

Motor response is not instantaneous. Rotor inertia, electrical characteristics, propeller loading, battery voltage, and controller behavior influence how quickly thrust changes after a command. The flight controller's design must account for this response when regulating attitude and movement. Incorrect assumptions about actuator dynamics can cause sluggish control or instability.

Propulsion also imposes hard physical limits. A motor cannot produce unlimited thrust, and a battery cannot deliver unlimited current. When the aircraft approaches maximum output, the controller may have reduced authority to correct disturbances. An aircraft carrying a heavy payload or operating under demanding conditions may have less reserve capacity than expected. Control allocation should therefore account for the actual operating limits rather than assume that every requested correction is achievable.

Power distribution connects the battery to the propulsion system, flight controller, sensors, communication equipment, and other electrical loads. Some systems use separate regulated supplies for sensitive electronics, while others integrate several power functions on a common board. The arrangement influences efficiency, thermal performance, voltage stability, and the consequences of an electrical fault.

Voltage regulators convert the battery's varying voltage into levels suitable for individual components. Their performance must be sufficient for expected load changes, startup conditions, and transient demands. A regulator that becomes unstable under a sudden increase in current may cause processor resets or sensor interruptions. A power design should therefore be assessed under realistic electrical loads rather than only with the aircraft idle.

Battery monitoring provides information about remaining energy and electrical health. Depending on the hardware, the system may measure voltage and current directly or estimate battery condition using a model. Neither method guarantees a perfect prediction of remaining flight time. Battery aging, temperature, discharge rate, internal resistance, and load changes influence the relationship between measured values and available energy.

Electrical noise can affect both propulsion electronics and sensitive sensors. Rapid switching currents, long conductors, poor grounding, and inadequate filtering may introduce disturbances into measurement or communication circuits. The physical layout should minimize unwanted coupling and provide appropriate separation between high-current paths and sensitive signals. Mechanical connections must also remain secure under vibration.

From a security perspective, the propulsion system is safety-critical because it converts electronic commands into physical movement. Unauthorized changes to motor-control parameters, unvalidated actuator commands, or compromised firmware may affect how the aircraft responds. However, such risks should be evaluated against the actual interface and implementation rather than assumed from the mere presence of an electronic speed controller.

A secure design restricts access to critical configuration settings, validates commands, and monitors relevant fault information. Where telemetry is available, unexpected discrepancies between

commanded and observed behavior may help identify a malfunction. Such discrepancies do not establish malicious activity by themselves; damaged hardware, electrical problems, or ordinary control faults can produce similar symptoms.

Testing should be performed with suitable safeguards. Propulsion systems can cause serious injury when motors start unexpectedly, even when the aircraft is not airborne. Bench inspection and software testing should follow appropriate equipment procedures, and experiments that do not require motor rotation should be performed with propulsion made safe. Dynamic tests should use controlled facilities and equipment appropriate to the aircraft.

## Building a Hardware Inventory and Identifying Security-Critical Components

A hardware inventory provides the factual foundation for understanding what an aircraft contains, how its components interact, and which elements deserve particular security attention. Without an accurate inventory, an assessment may overlook an exposed interface, misunderstand a communication dependency, or assume that a device contains protections that are absent from the installed hardware revision.

The inventory should identify the major computing components, including flight controllers, companion computers, communication modules, navigation receivers, sensor assemblies, electronic speed controllers, power-management boards, and payload-processing hardware. For each component, the record should include its manufacturer or supplier, model designation, hardware revision where available, firmware version, role, and relevant interfaces. These details help distinguish components that look similar but differ in capabilities or security configuration.

The inventory should also describe relationships between components. A flight controller may receive data from an inertial sensor over a local bus, communicate with a ground station through a telemetry module, and exchange higher-level commands with a companion computer. Mapping these connections helps identify where information crosses trust boundaries and where a failure could affect several functions.

Hardware identification should rely on reliable evidence. Board markings, documented configuration records, approved bills of materials, firmware information, and manufacturer documentation can help establish component identity. Visual inspection alone may not reveal the precise processor variant or whether a board has been modified. Where uncertainty remains, the inventory should record that uncertainty instead of presenting an unverified assumption as fact.

Version tracking is especially important. A board revision may alter a connector assignment, replace a component, change a power circuit, or introduce a different processor. Firmware built for one revision may not be appropriate for another. An accurate record allows engineers to associate test results and known limitations with the specific configuration that was evaluated.

Security-critical components should be identified according to the consequences of their compromise or failure. The flight controller usually deserves high priority because it coordinates stabilization and actuator commands. Firmware storage and boot mechanisms are important because they influence which code executes. Communication modules matter because they connect the aircraft to operators and external services. Navigation sensors influence position and orientation estimates, while power-management components affect the availability of the entire computing system.

Companion computers can also be security-critical when they process sensitive data or supply mission commands. Their importance depends on their privileges and integration with the flight controller. A computer used only to store non-sensitive payload records may have a different risk profile from one that can modify mission settings or communicate with multiple aircraft. Inventory records should describe these permissions and dependencies, not merely list the installed processor.

Debug and maintenance interfaces should be documented alongside ordinary operational connections. Their presence, intended purpose, access restrictions, and maintenance requirements should be recorded. This helps determine whether diagnostic access is appropriately controlled and whether legitimate servicing can be performed without weakening security unnecessarily.

Power dependencies deserve explicit attention because they can affect multiple components simultaneously. A shared regulator may supply the flight controller and several sensors, meaning that one electrical fault can degrade several apparently independent functions. A communication module may reset when propulsion loads create a voltage disturbance. Recording these relationships helps engineers identify common-cause failures that would otherwise be missed.

The inventory should extend to firmware and supporting software where possible. Hardware and software cannot be assessed independently when the security of a component depends on boot settings, driver behavior, update mechanisms, or configuration parameters. Recording firmware versions, software dependencies, secure-boot status where verifiable, and update procedures provides a more complete view of the system.

A useful inventory also distinguishes between observed facts and assumptions. For example, the presence of a cryptographic

accelerator does not prove that the aircraft uses strong authentication. A processor's secure-boot capability does not prove that secure boot is enabled. A connector described as a maintenance port does not establish that it is protected against unauthorized access. These properties require verification through documentation, configuration review, or authorized testing.

The inventory should support lifecycle management rather than serve as a document created once and then forgotten. Component replacements, firmware upgrades, changes to payload equipment, and modifications to communication interfaces can alter the system's security characteristics. Configuration changes should be recorded, reviewed, and validated before the modified aircraft returns to operational service.

For fleet operators, consistent inventory practices allow comparisons across aircraft and help identify configurations that have fallen behind approved versions. They can also support incident response by establishing which devices share a component, firmware release, or configuration associated with a newly discovered weakness. Such information improves the ability to determine the scope of a problem without assuming that every aircraft is affected equally.

A hardware inventory becomes most valuable when it connects physical components to their functions, privileges, dependencies, and maintenance status. That record allows engineers to determine where security controls are needed, which interfaces require closer examination, and how a hardware or firmware change might influence the aircraft's behavior. It also creates a dependable baseline against which future assessments can measure whether the system remains consistent with its documented design.

# Constructing a Drone Cybersecurity Laboratory

## Selecting Compatible Autopilot Software and Simulation Platforms

A well-designed drone cybersecurity laboratory provides a controlled environment in which the software, communications, computing architecture, and safety mechanisms of unmanned aircraft can be examined without exposing an operational aircraft to unnecessary risk. Its purpose extends beyond learning how an autopilot works. It allows researchers to investigate how flight-control software responds to unexpected inputs, how communication interfaces handle malformed or inconsistent data, how configuration errors affect system behavior, and how security controls perform when normal operating assumptions no longer hold.

The most effective laboratory begins with a clear understanding of the aircraft systems being studied. An unmanned aircraft is not a single software application but a collection of interdependent components. The autopilot processes sensor information and generates control commands. A simulation engine models the physical behavior of the aircraft. A ground control application provides mission-planning and monitoring functions. Communication interfaces connect these components, while supporting tools record events and help researchers analyze results. A weakness or incompatibility in any of these elements can affect the validity of an experiment.

Selecting compatible software is therefore an engineering decision rather than a matter of installing several popular tools and expecting them to operate together. Researchers must consider processor architecture, operating-system support, software dependencies, communication protocols, simulation interfaces, and the capabilities of the selected aircraft model. They must also determine whether the software versions are sufficiently stable for repeatable experiments and whether their documentation describes the expected configuration.

Autopilot software generally provides functions such as sensor processing, state estimation, flight stabilization, navigation, mission execution, communication handling, and failsafe management. Some systems are designed primarily for microcontrollers, while others can also operate in software simulations or on development computers. A laboratory should distinguish between software intended to run on actual flight-controller hardware and software builds intended for simulation. Although these builds may share much of their source code, their execution environments and hardware dependencies can differ substantially.

Software-in-the-loop simulation is particularly useful during the early stages of laboratory construction. In this arrangement, the autopilot software runs as a computer process, while a separate simulation environment models the aircraft and its surrounding physical conditions. The autopilot receives simulated sensor measurements and produces control outputs that influence the simulated aircraft. This arrangement allows researchers to investigate control behavior, navigation logic, communication handling, and selected failure conditions without powering real motors or placing a physical aircraft at risk.

Hardware-in-the-loop simulation introduces real flight-control hardware into the test arrangement. The physical controller executes its firmware while receiving simulated sensor information and

exchanging actuator commands with the simulation system. This approach can reveal issues associated with actual processor timing, hardware interfaces, embedded configuration, and device-specific behavior that a computer-based simulation may not reproduce accurately.

Neither approach replaces the other. Software-in-the-loop testing is generally easier to scale and automate, while hardware-in-the-loop testing provides greater exposure to the behavior of real embedded components. A mature research program uses each method for the questions it can answer and recognizes the limitations of its results.

The simulation platform should be selected according to the research objectives. A laboratory focused on basic stabilization may require only a suitable aircraft model, an autopilot simulation build, and a way to inspect telemetry. Research into autonomous navigation may require more detailed environmental models, simulated positioning data, and mission-planning capabilities. Investigations of communication reliability may need configurable link delays, message-loss conditions, and tools for recording the interaction between software components.

The aircraft model must also match the intended study. A multirotor model does not reproduce every relevant property of a fixed-wing aircraft. Their propulsion systems, aerodynamic behavior, control authority, and responses to disturbances differ. Hybrid aircraft introduce additional complexity because they transition between operating regimes. Selecting an unsuitable model can lead researchers to draw conclusions that do not apply to the aircraft they intend to understand.

Compatibility should be established through documented support rather than assumption. The autopilot version, simulation engine, ground control application, and communication libraries may each evolve independently. An interface that worked in one release may behave differently after an update. Before beginning an experiment,

the researcher should establish a known-compatible combination and record the versions of all relevant components.

Software provenance matters as well. Research environments should use legitimate, documented software distributions and retain enough information to identify the origin of each installed component. Where source code is available, its release identifier and any local modifications should be recorded. When prebuilt packages are used, the package version and source should be documented. This practice helps researchers distinguish an upstream software defect from a local modification or an installation problem.

The laboratory should also separate experimentation from uncontrolled software changes. Automatically updating every component may introduce new variables into a test program. A better approach is to maintain a stable baseline and evaluate updates in a separate working copy before incorporating them into the main environment. Security patches should not be ignored; instead, their effects should be assessed deliberately, with the relevant version changes recorded.

Documentation quality is another selection criterion. A platform with clearly explained installation requirements, simulation interfaces, supported aircraft models, configuration options, and diagnostic procedures is easier to evaluate scientifically. Researchers should understand how the simulator represents sensor noise, actuator limits, environmental disturbances, and timing. A simulation that presents idealized measurements may be appropriate for some control experiments but insufficient for investigations that depend on realistic sensor uncertainty.

The laboratory should establish a defined scope before selecting its software. That scope may include validating communication behavior, evaluating failsafe logic, studying configuration integrity, examining software update mechanisms, or measuring the effects of sensor faults. Defining the intended questions helps avoid

unnecessary complexity and ensures that the selected tools provide the capabilities required to produce meaningful evidence.

## Configuring an Isolated Development and Testing Environment

Isolation is one of the most important properties of a drone cybersecurity laboratory. A simulation may behave like an aircraft in many respects, but the software running inside it can still communicate with other systems through network interfaces, local services, files, and shared computing resources. If those connections are not understood and controlled, a test intended for a simulated aircraft may affect unrelated systems or expose the laboratory to unnecessary risk.

The environment should be designed so that experimental traffic remains separate from ordinary business, personal, and production networks. A dedicated virtual network, isolated test machine, or appropriately configured virtual machine can provide a boundary between laboratory activity and unrelated computing resources. The suitable arrangement depends on the host system, the protocols under examination, and the degree of isolation required.

Virtual machines are useful because they provide a distinct operating-system environment with configurable virtual hardware and network connections. A researcher can maintain one environment for autopilot development and another for analysis tools, reducing the chance that dependencies from one project interfere with another. Snapshots can preserve a known state before a change, allowing the researcher to return to a documented baseline when an experiment produces unexpected results.

Containers can provide a lighter way to separate applications and their dependencies. They are particularly useful when the autopilot, simulator, or supporting services can run in a containerized environment. However, containers share the host operating-system kernel and do not provide the same isolation properties as a separate virtual machine in every configuration. Their suitability depends on the application's requirements and the security boundary being established.

The network design should reflect the laboratory's actual communication needs. The autopilot simulator may exchange telemetry with a ground control application, while a separate analysis process collects logs. These connections can be permitted within a dedicated test network without granting the experimental systems unrestricted access to external services. Where internet access is needed to retrieve documentation or software updates, it should be separated from the experiment itself whenever practical.

Network isolation does not mean that every component must be disconnected from every other component. It means that each connection should have a defined purpose and that unnecessary paths should be removed. A laboratory may need communication between a simulated flight controller and a ground station, but it generally does not need the experimental process to reach unrelated devices on a home or institutional network. Clear boundaries reduce accidental exposure and make the experimental arrangement easier to document.

Researchers should pay attention to how simulation tools bind to network interfaces. Some applications listen on all available interfaces by default, while others expose services only on a local interface. A service intended for communication between two laboratory processes should not be assumed to be private merely because it runs on a research computer. Its actual listening address, port, and access controls should be verified.

Physical isolation may also be appropriate when experiments involve real radios, actual flight-controller boards, or other devices capable of transmitting or affecting physical equipment. Radio-frequency experimentation must comply with applicable spectrum regulations and should use suitable controlled arrangements. For most initial software-security experiments, simulated communication is sufficient and avoids the additional complexity of physical radio testing.

The host operating system should be maintained with appropriate security updates, and laboratory software should run with only the privileges it requires. Development tools sometimes request elevated permissions for device access, debugging, or system configuration. Those permissions should be granted only when necessary and should not become the default for routine simulation. Restricting privileges reduces the consequences of software defects and helps contain accidental changes.

Secrets and credentials require separate consideration. Some research environments use signing keys, access tokens, test certificates, or credentials for supporting services. These should not be embedded unnecessarily in source code, stored in public repositories, or included in experiment logs. A laboratory should use test-specific credentials and protect any keys that have operational significance. If a credential is needed only for a temporary experiment, its lifetime and access should be limited accordingly.

Filesystem organization contributes to reproducibility and safety. Source code, simulator models, configuration files, logs, experimental notes, and generated results should have clearly defined locations. Read-only copies of baseline configurations help prevent accidental modification. Separate working directories allow researchers to experiment without altering the original material required to reproduce earlier tests.

The environment should also provide a clear method for restoring a known state. A repeatable reset procedure may involve reverting a virtual-machine snapshot, restoring a configuration file, rebuilding a software image, or reinitializing the simulator. Resetting should remove unintended state left by previous experiments, such as changed parameters, accumulated logs that affect storage availability, or processes that remain active after a test.

Isolation must be verified rather than assumed. Researchers should inspect the environment's network configuration, identify active services, confirm which resources are shared, and verify that the permitted communication paths correspond to the intended design. The purpose is not simply to create a separate desktop workspace but to understand the actual boundaries between the simulated aircraft, the supporting applications, and the host system.

A documented laboratory baseline should describe the operating system, software versions, network arrangement, resource allocation, permissions, and reset procedure. When a test produces an unexpected result, this record helps establish whether the cause lies in the software under examination or in an unnoticed change to the environment.

# Running Software-in-the-Loop Simulations of Unmanned Aircraft

Software-in-the-loop simulation provides a controlled method for evaluating autopilot behavior before introducing real flight hardware. It combines executable flight-control software with a mathematical representation of the aircraft's physical behavior. The simulator calculates how the aircraft responds to control inputs and environmental conditions, then supplies simulated sensor

information to the autopilot. The autopilot processes those measurements and returns control outputs, completing the feedback loop.

The quality of the experiment depends on the fidelity of this interaction. The simulation must represent the relevant physical properties well enough to support the question being investigated. A simplified model may be entirely adequate for checking whether a communication interface accepts a particular message format. The same model may be inadequate for evaluating a control algorithm's response to turbulence, actuator saturation, or aerodynamic instability.

A typical simulation contains several interacting elements. The vehicle model represents mass, inertia, propulsion, aerodynamic behavior, and control surfaces as appropriate. The environment provides conditions such as gravity, wind, and other disturbances. Sensor models generate measurements that resemble the output of the aircraft's sensors. The autopilot interprets those measurements, estimates the aircraft's state, and calculates control commands. The simulator then applies those commands to the vehicle model.

The relationship between simulation time and software execution time requires attention. Some simulators attempt to advance the model in real time, while others can run faster or slower depending on computing resources and configuration. If the simulator advances physical time inconsistently or the autopilot receives measurements with unrealistic timing, the resulting behavior may not represent the conditions of an actual aircraft.

Researchers should establish the expected timing behavior of their chosen platform. They should understand how sensor updates are scheduled, how actuator commands are applied, and how the simulator handles delays or missed updates. When timing is relevant to the experiment, the test should record both simulation time and

appropriate wall-clock timing information. This distinction helps prevent misleading interpretations of apparent performance.

The first validation stage should establish that the baseline simulation behaves as expected. The aircraft should initialize correctly, the autopilot should enter the intended mode, and the simulator should report plausible state information. Basic commands can then be used to verify that the control loop responds in a manner consistent with the selected aircraft model and configuration.

These initial checks are not merely preliminary housekeeping. They establish a reference against which later experiments can be compared. If the baseline already contains unexplained oscillations, incorrect sensor readings, or repeated mode changes, a later security test cannot reliably attribute those behaviors to the condition being studied.

Once the baseline is stable, the researcher can examine one variable at a time. A study might evaluate how the autopilot responds to delayed sensor measurements, temporary navigation-data loss, a communication interruption, or an unexpected but valid state transition. The experimental condition should be introduced through a controlled simulation feature or a documented test harness, rather than through uncontrolled changes that make the outcome difficult to interpret.

Sensor-fault experiments should distinguish between missing data and misleading data. A missing measurement may be detected through a timeout, while an incorrect measurement may arrive on schedule and pass basic format checks. These conditions test different assumptions in the estimation and control architecture. The experiment should record whether the autopilot detects the problem, how it represents uncertainty, whether it changes operating modes, and whether its response remains within the intended safety limits.

Communication testing can examine whether messages are accepted, rejected, delayed, or ignored under defined conditions. Researchers should establish what constitutes valid behavior for the interface before conducting the test. An application may legitimately reject an unfamiliar message or ignore a command that is not permitted in the current flight mode. Such behavior should not be classified as a defect merely because the requested action did not occur.

Fault injection should be bounded by the purpose of the experiment. The objective is to observe system behavior under controlled conditions, not to create an uncontrolled environment where failures interact unpredictably. Where multiple faults are relevant, they can be introduced separately first, then combined in a carefully documented sequence. This approach helps identify which conditions contribute to a result and which interactions deserve additional investigation.

Simulation models have limitations that must be acknowledged. A software simulation may not reproduce actual electrical noise, physical connector faults, processor-specific timing, motor-controller behavior, or every sensor characteristic. Some models approximate aerodynamic effects and environmental conditions. A successful simulation therefore provides evidence about the modeled system, not automatic proof that an identical outcome will occur on real hardware.

Model validation improves confidence in the results. Researchers can compare simulated behavior with documented specifications, previously established test results, or carefully collected measurements from authorized and controlled experiments. Differences should be recorded and explained where possible. If the model omits a feature that is central to the research question, the experiment should be redesigned or its conclusions appropriately limited.

Simulation is particularly valuable for evaluating rare or hazardous conditions. It can reproduce navigation loss, extreme battery estimates, unexpected mode transitions, or multiple sensor faults without requiring an aircraft to encounter those conditions during live flight. This ability makes it possible to examine how the software handles unusual events that would be difficult, expensive, or unsafe to create in a physical environment.

The researcher should preserve the complete experimental configuration. This includes the autopilot build, simulator version, vehicle model, relevant parameters, initial conditions, fault settings, and output logs. If the experiment depends on a random process, such as simulated sensor noise, the random seed or equivalent reproducibility mechanism should be recorded where supported. Otherwise, two runs may differ for reasons unrelated to the software change being evaluated.

Results should be interpreted against predefined expectations. A successful run is not necessarily one in which the aircraft completes its simulated mission. It may instead be one in which the system correctly rejects an invalid request, enters a documented degraded mode, or records an error without affecting unrelated functions. Defining success in advance prevents researchers from judging the same behavior differently after seeing the outcome.

# Integrating Ground Control Applications With Simulated Flight Systems

Ground control applications provide the interface through which operators and researchers monitor an aircraft, inspect its status, configure permitted functions, and manage missions. In a cybersecurity laboratory, the ground station also serves as an

important observation point. It can display telemetry, report mode changes, show navigation estimates, and expose the practical consequences of communication or configuration problems.

Integration requires a clear understanding of the communication path between the ground control application and the simulated flight system. The ground station may communicate directly with the simulated autopilot, or the architecture may include an intermediate service that routes messages between components. Each arrangement has different implications for troubleshooting, access control, timing, and evidence collection.

The communication protocol defines how messages represent commands, telemetry, status information, and other data. However, a protocol specification does not necessarily determine the complete behavior of a particular implementation. Applications may support different message subsets, apply different validation rules, or interpret optional fields differently. Compatibility should therefore be tested using the actual software versions selected for the laboratory.

The first integration objective is to establish ordinary communication. The ground station should connect to the intended simulated vehicle, receive status information, and display a coherent view of the aircraft's state. The autopilot should recognize only the expected command sources and should respond according to its documented mode and safety restrictions. If the ground station reports a connection but telemetry remains absent, researchers should determine whether the issue concerns message routing, initialization, protocol compatibility, or the simulation itself.

Command and telemetry paths should be understood separately. Telemetry generally communicates information about the aircraft's estimated condition and system status, while commands request changes to behavior or configuration. A connection that permits telemetry does not necessarily need unrestricted command

authority. The laboratory should preserve this distinction wherever the architecture supports separate permissions or interfaces.

Message validation is especially important when ground control software accepts data from several sources. The receiving component should check message structure, relevant field values, expected timing, and the authority of the sender where appropriate. A syntactically correct message may still be invalid in the current operating state. For example, a command may be inappropriate when navigation information is unavailable or when the aircraft is already executing a safety response.

The system should also handle stale information. A telemetry display may continue showing the last known position even after updates have stopped, potentially giving the impression that the aircraft's state is current. A well-designed application should make communication status and data freshness clear. For laboratory analysis, timestamps and connection-state events are valuable because they help distinguish a genuine state change from a display that has stopped updating.

Ground control applications often provide configuration and mission-management features. These capabilities should be evaluated in relation to the permissions available to the operator and the safeguards enforced by the autopilot. A user interface may prevent an operator from selecting an invalid value, but the autopilot should not rely exclusively on that interface for protection. Commands can originate from other clients or be generated by software, so critical restrictions should be enforced at the component responsible for the affected function.

Integration testing should examine normal operations as well as expected failures. Researchers can verify whether telemetry resumes after a temporary connection interruption, whether the application reports invalid data appropriately, and whether mode changes remain synchronized between the ground station and the simulated

aircraft. The tests should also establish what happens when the ground station and autopilot disagree about the current mode or the completion of an operation.

Time synchronization can affect the interpretation of results. Logs produced by separate processes may use different clocks or timestamp conventions. If those records are compared without accounting for the difference, a researcher may infer the wrong order of events. The laboratory should use a consistent time basis where practical and document any known offsets or timing limitations.

Ground station logs should be collected without allowing the logging process to overwhelm the simulation or distort timing. Excessive logging can consume storage, increase processing load, or obscure important events among large quantities of routine data. Researchers should identify the records required for the experiment and configure the environment to preserve those records in a controlled manner.

Where several ground control applications or clients are tested, their authority and interaction should be documented. Multiple clients may display the same telemetry but have different capabilities to change parameters or request flight actions. If the system permits simultaneous command sources, the test should establish how competing requests are handled and which component has final authority.

A successful integration is not defined simply by whether the ground station displays an aircraft icon. It requires evidence that the expected messages are exchanged, commands are interpreted according to documented rules, invalid requests are handled appropriately, and the system's status remains understandable during interruptions. The integration should be repeatable, with the same configuration producing comparable behavior across successive runs.

# Designing Repeatable Experiments, Recording Evidence, and Validating Results

A cybersecurity laboratory becomes scientifically useful when its experiments can be repeated and their results independently understood. An observation that cannot be reproduced may still be informative, but it is difficult to distinguish a genuine software weakness from an environmental condition, an installation error, or a transient fault. Repeatability requires a defined procedure, a known baseline, controlled variables, and sufficient evidence to reconstruct what occurred.

Every experiment should begin with a precise research question. Instead of asking whether an autopilot is secure in general, a researcher might ask whether a specific software build detects delayed sensor data under a defined simulation condition. Another experiment might examine whether an invalid configuration value is rejected, whether communication loss activates the expected failsafe, or whether a ground station clearly identifies stale telemetry.

The question determines the required setup and evidence. An investigation of sensor validation may require autopilot logs, simulator records, and timestamps for injected measurements. An investigation of mode transitions may require records of the initial mode, the triggering event, the requested transition, and the resulting state. Collecting unrelated data without a clear purpose can make analysis more difficult and increase the chance that the relevant evidence is overlooked.

A baseline run should be completed before introducing the experimental condition. This run establishes normal behavior under

the selected configuration. It should confirm that the simulator initializes correctly, the autopilot operates in the intended mode, communication is functioning, and the required telemetry is available. Any unexplained anomaly should be investigated before the experiment proceeds.

The experimental procedure should then describe how the condition is introduced, how long the observation lasts, and what constitutes a successful or unsuccessful result. Relevant parameters should be specified explicitly. Terms such as short delay, high load, or abnormal sensor reading are not sufficiently precise for a reproducible experiment unless they are tied to measurable values or clearly defined conditions.

Only the variables necessary for the test should be changed. If a researcher simultaneously changes the autopilot version, simulator model, sensor configuration, and communication settings, the resulting behavior may have several plausible explanations. Changing one principal variable at a time is often the clearest initial method. More complex experiments can examine interactions between variables after the individual effects are understood.

Evidence should be collected from the components that can establish the relevant facts. Autopilot logs may show state transitions and rejected commands. Simulator records may show the condition that was introduced and the resulting vehicle response. Ground control logs may establish what the operator interface displayed. System-level records may reveal process failures or resource exhaustion. No single log source necessarily provides the complete picture.

Timestamps should be preserved carefully. A useful record identifies when the experimental condition was introduced, when the relevant component received it, when the system detected or responded to it, and when the resulting behavior became observable. The available software may not provide all these timestamps directly, so researchers should document the limits of the timing evidence rather

than imply a level of precision that the instrumentation cannot support.

Raw evidence should be preserved separately from derived analysis. Original log files, configuration records, and test outputs should remain unchanged once collected. Analysis scripts can then produce filtered views, event timelines, graphs, or statistical summaries without replacing the source material. This separation makes it possible to verify whether an interpretation follows from the original observations.

Where evidence integrity matters, researchers can calculate cryptographic hashes of collected files and record the results alongside the experiment identifier. A hash can help determine whether a file has changed since it was recorded, although it does not by itself prove that the original data was accurate or that the collection process was trustworthy. Evidence integrity depends on both the protection of the records and the reliability of the method used to obtain them.

The experiment record should identify the software versions, operating environment, vehicle model, configuration files, relevant parameters, and any local modifications. It should also document unexpected events, including failed initialization, missing logs, resource constraints, and deviations from the planned procedure. Negative results and incomplete runs should not be discarded merely because they do not support the initial expectation. They may reveal important limitations in the test setup or in the software under examination.

Validation requires checking that the observed result is consistent with the evidence and that alternative explanations have been considered. If a communication interruption is followed by a mode change, the researcher should establish whether the interruption triggered that change or whether another event occurred at the same

time. Comparing autopilot logs, simulator events, and ground station records can help establish the sequence.

Repeated trials provide additional confidence. If the same condition produces the same outcome across several runs, the result is less likely to depend on an accidental event. When outcomes vary, the variation should be investigated. It may reflect random sensor noise, nondeterministic scheduling, changing initial conditions, incomplete reset procedures, or a genuine defect that occurs only under particular circumstances.

Statistical analysis may be appropriate when the experiment produces quantitative measurements, such as response latency, recovery time, or the frequency of a defined event. Researchers should report the number of trials, the measurement method, and the relevant variability. A single unusually fast or slow result should not be treated as representative without supporting evidence.

The laboratory should also distinguish between observed behavior and inferred security impact. Demonstrating that a message is accepted does not automatically establish that an attacker can control the aircraft. The researcher must determine what authority the message carries, what conditions are required to submit it, whether other safeguards intervene, and what consequences follow in the tested configuration. Conclusions should remain proportional to the evidence.

A useful report describes the affected component, the tested software and hardware configuration, the preconditions, the observed behavior, the expected behavior, and the practical limitations of the experiment. It should distinguish confirmed findings from hypotheses and identify the additional evidence needed to resolve uncertainties. This approach produces results that other engineers can evaluate without relying on unsupported claims.

Responsible research also requires attention to disclosure and operational boundaries. Experiments should remain within systems the researcher owns or is authorized to assess. Findings that may affect deployed aircraft should be handled through appropriate reporting and remediation processes. A simulated demonstration can establish a plausible technical weakness, but claims about real-world exposure should be supported by further evidence and a clear understanding of the affected implementations.

As the laboratory matures, its experiment records become a valuable technical resource. A collection of carefully documented tests can show how software versions behave under comparable conditions, whether a fix resolves a previously observed defect, and whether a change introduces regressions elsewhere. The resulting evidence supports disciplined engineering decisions and provides a defensible basis for evaluating the reliability and security of unmanned aircraft software.

# Autopilot Software and Firmware Security

## Understanding Firmware Structure, Boot Sequences, and Runtime Behavior

Autopilot software is responsible for translating information about an unmanned aircraft's condition into decisions that govern its movement, navigation, and response to changing circumstances. Its responsibilities extend beyond flight stabilization. Depending on the aircraft and its configuration, the software may process sensor measurements, estimate position and orientation, manage operating modes, execute mission instructions, communicate with a ground control station, monitor power conditions, and initiate protective responses when normal operation becomes unreliable. Because these functions depend on one another, firmware security must be examined as a system property rather than as an isolated question of whether a particular section of code contains a defect.

Firmware is the software stored in or deployed to an embedded computing device to provide its operational behavior. In an autopilot, it commonly includes the application logic, hardware-specific drivers, communication interfaces, scheduling mechanisms, control algorithms, and supporting libraries needed to operate the flight controller. Depending on the architecture, the firmware package may also incorporate bootloader components, configuration defaults, metadata, and information used to identify the software version. These elements differ across manufacturers, processors, and development platforms, but they share a common requirement: the

software must start reliably, interact correctly with hardware, and maintain predictable behavior throughout operation.

Understanding the structure of firmware begins with distinguishing the source code from the compiled software that executes on the target processor. Source code expresses the program in a human-readable language. A compiler transforms that code into machine instructions suitable for a particular processor architecture, while a linker combines compiled modules and resolves references between them. The resulting executable may be converted into a firmware image that can be stored in flash memory or another supported storage device. Depending on the development process, the final image may include metadata, initialized data, checksums, signatures, or other information used during installation and startup.

This distinction matters because the source repository does not necessarily represent the exact software running on an aircraft. A deployed image may have been built from a different revision, compiled with different options, or modified by a local development process. Some features may be enabled only for particular hardware targets, while others may be excluded to conserve memory or meet timing requirements. A reliable security assessment must therefore establish the identity of the software being examined and avoid assuming that a source-code finding automatically applies to every compiled variant.

The boot sequence is the chain of operations through which the flight controller moves from an unpowered or reset condition into a functioning state. In a typical embedded architecture, the processor begins execution at a defined location and performs initial operations required to establish a usable computing environment. Depending on the hardware, these may involve setting up memory, configuring clocks, initializing low-level processor facilities, and transferring control to a bootloader or application entry point.

A bootloader is a small program that prepares or loads the main firmware. Its responsibilities vary by implementation. Some bootloaders support firmware updates over a wired connection, while others can receive software through a designated communication interface. Certain systems use a bootloader that checks the integrity or authenticity of an image before permitting execution. Others rely on different combinations of hardware protections, update procedures, and software-level checks. The existence of a bootloader alone does not establish that the startup process is secure; the relevant question is whether each transition in the boot chain is adequately controlled.

Secure boot, where supported, attempts to ensure that the processor executes only software that satisfies a defined authenticity or integrity policy. A cryptographic signature can provide evidence that an image was approved by a holder of the corresponding signing key. A hash can identify changes to the data being checked, but a hash alone does not establish who produced the image. These mechanisms serve different purposes and should not be treated as interchangeable.

A secure startup design must also account for trust in the verification process itself. If an attacker can replace the verification code, alter its trusted configuration, or obtain an authorized signing key, the intended protection may be undermined. Hardware-supported roots of trust, protected key storage, controlled update authorization, and carefully designed recovery procedures can strengthen the chain of verification. However, the protections available depend on the processor and board architecture. A software feature described as secure boot may offer limited assurance if its initial verification stage is not itself protected.

The startup process must establish more than software authenticity. The autopilot also needs to determine whether essential hardware is available and whether critical subsystems can operate. It may

initialize inertial sensors, read calibration data, configure communication peripherals, establish scheduler services, and prepare control outputs. The order of these operations is important. Flight-control logic should not assume that a sensor is ready merely because the corresponding driver has been loaded.

A robust initialization process checks that required components are present and behaving within expected limits. If a critical sensor fails to respond, the system should avoid entering an operating mode that depends on that sensor without an appropriate alternative. The correct response depends on the aircraft design and the available redundancy. Some faults may permit limited operation, while others should prevent arming or require a transition to a protective state.

The handling of startup failures is particularly important because incomplete initialization can create conditions that differ from both normal operation and a complete system shutdown. A partially initialized communication interface may accept data before the application is ready to process it correctly. A failed sensor may produce default or stale values. An interrupted update may leave the system unable to start the new image. Secure design therefore considers not only the expected startup sequence but also what happens when individual stages fail or are interrupted.

Once startup is complete, runtime behavior is shaped by the firmware's scheduling model. Many autopilots execute multiple tasks with different timing requirements. Fast control loops may run at a relatively high frequency, while telemetry, mission management, logging, and maintenance functions may operate less frequently. An embedded scheduler or real-time operating system coordinates these activities, helping ensure that time-critical work receives the processing time it requires.

The separation of tasks introduces security and reliability considerations. A computationally expensive operation in a noncritical task can interfere with other functions if resource limits

are not enforced. A communication handler that accepts excessive input may consume memory or processing time needed by a control loop. An unexpected failure in a supporting service may affect other components if error isolation is weak. The design should therefore limit resource consumption, establish clear task responsibilities, and provide appropriate mechanisms for detecting missed deadlines or abnormal task behavior.

Runtime security also depends on the relationships between components. The navigation subsystem may supply state estimates to the flight controller, the mission manager may request a change in operating mode, and the communication layer may receive commands from an external application. Each component must have a clear understanding of which data it can trust and which actions it is permitted to initiate. A message that is valid at the protocol level may still be unsuitable for the current flight state. Validation must account for meaning and context, not merely format.

A useful firmware assessment follows the system from startup through normal operation and into failure handling. It examines which components establish trust, when configuration values become active, how sensor readiness is determined, how operating modes change, and what the software does when essential assumptions are no longer satisfied. This approach reveals weaknesses that may not be visible when individual functions are examined without their surrounding execution context.

# Exploring Open-Source Autopilot Code and Embedded Software Dependencies

Open-source autopilot software provides researchers with an opportunity to study implementation details, follow the flow of control through the code, and compare design decisions against

documented requirements. Source availability makes it possible to examine how a system processes incoming messages, manages configuration, handles errors, and interacts with hardware. It does not, however, guarantee that the software is free from defects or that every part of the deployed system is publicly available.

A source repository typically contains more than the code that directly controls the aircraft. It may include hardware abstraction layers, peripheral drivers, communication libraries, mathematical routines, mission-management components, configuration definitions, build scripts, automated tests, documentation, and simulation interfaces. The repository may support several processor families and board designs, with conditional compilation determining which components are included in a particular firmware build.

The first task in reviewing such a repository is to establish its structure and intended use. Documentation and build configuration can identify the principal application modules, supported hardware targets, external libraries, and testing arrangements. Understanding these boundaries helps the researcher identify where security-sensitive operations occur and which components provide shared services to the rest of the system.

Hardware abstraction layers are particularly important in embedded software. They provide interfaces that allow higher-level application code to use hardware functions without depending directly on every processor-specific detail. A sensor driver, for example, may obtain measurements through a standardized interface even though the underlying communication bus differs between boards. This arrangement improves portability, but it also means that a defect in a shared abstraction or driver can affect multiple higher-level functions.

Communication libraries deserve similar attention. They may parse messages, encode outgoing data, manage framing, or provide

transport-independent interfaces. Their behavior can influence the security of every application component that relies on them. A parser that handles message lengths incorrectly, for instance, may expose multiple calling functions to the same underlying weakness. Conversely, a well-designed shared validation layer can provide consistent protections across several interfaces.

The repository should be examined with attention to trust boundaries. A trust boundary exists where data, commands, or authority move between components that have different levels of trust. Examples include a communication interface receiving messages from a ground station, a mission manager interpreting externally supplied mission data, or a firmware update mechanism accepting a new image. Data crossing such boundaries should be validated before it influences security-sensitive operations.

Researchers should trace important inputs beyond their initial reception. A message parser may correctly identify a field as an integer, yet later code may interpret that value as a buffer length, actuator limit, mode identifier, or memory index. Security depends on the entire processing path. Validation performed at the first interface may become insufficient if the value is transformed, combined with other values, or used under different assumptions later in execution.

Embedded dependencies introduce another layer of risk. Modern firmware may incorporate external cryptographic libraries, compression routines, mathematical libraries, communication stacks, storage systems, and operating-system components. Each dependency has its own development history, release schedule, configuration options, and possible vulnerabilities. The security of the completed firmware therefore depends partly on the properties of software maintained outside the main autopilot project.

A dependency inventory should identify the libraries included in the build, their versions, and the roles they perform. Where available, a

software bill of materials can provide a structured record of components and relationships. Such an inventory helps researchers determine whether a reported vulnerability affects the actual firmware configuration rather than merely appearing in a library that happens to exist somewhere in the repository.

Version identification requires care. A project may include a modified copy of a library rather than an unchanged upstream release. A dependency may be downloaded during the build, included as a source submodule, or incorporated directly into the repository. The version displayed in a project manifest may not accurately describe local modifications. A sound assessment records the specific source revision and checks whether the relevant code differs from the upstream component.

The presence of a known vulnerability does not automatically establish that the deployed autopilot is exploitable. The affected function may not be compiled into the firmware, the vulnerable code path may be unreachable, or the system may apply additional validation that prevents the unsafe condition. On the other hand, a dependency may have been modified in a way that introduces a weakness not covered by the upstream advisory. Assessment must therefore connect vulnerability information to the actual build and execution path.

Build scripts are part of the security boundary because they determine how source code becomes an executable image. A build system may select compiler versions, define feature flags, include target-specific files, download dependencies, and generate version metadata. If these steps are poorly controlled, two developers may produce different binaries from what appears to be the same source revision.

Reproducible builds can improve confidence by allowing independent parties to compare outputs generated from a documented source state and build environment. Perfect

reproducibility may be difficult where timestamps, toolchain differences, or nondeterministic build steps affect the output, but the goal remains valuable. At minimum, the build process should record its inputs and provide a way to explain significant differences between generated artifacts.

Code review should focus on the components most relevant to the system's threat model. This includes firmware update verification, message parsing, access control, persistent configuration, sensor-data handling, and functions that determine whether the aircraft may enter a particular operating mode. A review that treats every file as equally important can consume considerable effort without addressing the most consequential risks.

Commit history and issue discussions may also help explain why a particular design exists. A function that appears unusual may be compensating for a hardware limitation or preserving compatibility with an older interface. Understanding that context helps reviewers distinguish an intentional constraint from an implementation mistake. Nevertheless, historical rationale should not be accepted as proof of current security. The design must still satisfy its present requirements.

Open-source review is most effective when it combines code comprehension with evidence from tests, build configuration, documentation, and runtime observations. Source code reveals what the implementation is intended to do and how it is written. Build records show which portions are included. Dynamic testing demonstrates how the compiled program behaves under selected conditions. Together, these sources provide a stronger basis for conclusions than any one of them can provide independently.

# Examining Configuration Data, Build Artifacts, and Software Components

Configuration data determines how an autopilot behaves within the capabilities of its software and hardware. It can influence sensor calibration, control limits, communication settings, mission behavior, failsafe thresholds, and the availability of particular features. A configuration error may produce behavior that resembles a software defect, while an insecure configuration can weaken protections that the software otherwise supports.

Configuration values may originate from several locations. Some are compiled into the firmware as defaults, while others are stored in nonvolatile memory and retained across restarts. Additional settings may be loaded from a file, supplied by a ground control application, or generated during installation. The system may also calculate internal parameters at runtime from sensor measurements or other operating conditions.

These sources create questions of precedence and consistency. If a parameter exists both as a compiled default and as a stored value, the firmware must define which value takes effect. If a configuration migration changes the meaning or representation of a parameter, the software must handle older stored values correctly. An incomplete migration may leave the aircraft using an unexpected setting or cause startup to fail.

A configuration review should establish where each security-relevant value originates, who or what can modify it, how the modification is validated, and when the change becomes active. The review should also examine whether the value persists across reboot and whether it can be restored to a known safe state. These details are particularly

important for settings that affect arming, navigation, communication permissions, or emergency behavior.

Parameter validation should be performed at the point where values become authoritative. A graphical interface may restrict an operator to an approved range, but the firmware should not rely solely on the interface to enforce that restriction. Configuration may be introduced through another application, a file import, a maintenance utility, or a software interface. Critical constraints should therefore be checked by the component that applies the setting.

Validation must also consider relationships between values. Two individually acceptable parameters may form an invalid combination. A timeout may be shorter than the update interval of the process it monitors, or a configured operating limit may conflict with a sensor's supported range. Where such relationships matter, the software should validate the combined configuration and report errors clearly.

Persistent storage creates additional concerns. Power loss during a configuration write may leave a value partially updated, while memory wear or storage corruption can affect retained data. Robust designs use suitable storage procedures, integrity checks, and recovery behavior. When a configuration cannot be trusted, the autopilot should avoid silently treating uncertain data as valid if doing so could compromise safe operation.

Build artifacts provide another source of evidence. These include executable images, object files, symbol information, generated headers, configuration manifests, and other files produced during compilation. They can help establish which features were included, which target was built, and whether the delivered firmware corresponds to the source under review.

The compiler and linker influence the final behavior. Optimization settings can change instruction ordering and remove code paths that are not considered reachable. Conditional compilation can exclude entire features. Linker scripts determine where code and data are placed in memory. These choices can affect memory usage, startup behavior, timing, and the practical consequences of certain defects.

An artifact review should confirm that the build target matches the intended board and processor. Firmware compiled for an incorrect memory layout or peripheral configuration may fail to initialize correctly or interact with hardware in unexpected ways. A successful compilation is not proof that the resulting image is suitable for the intended device.

Binary metadata can reveal version identifiers, target information, build options, and other useful details, depending on how the software is packaged. Debug symbols may provide function names and source-level information that make analysis easier, while stripped binaries may expose less descriptive information. The absence of debug symbols does not make a binary secure; it merely changes the difficulty of understanding it.

Firmware authenticity and integrity should be assessed separately. A cryptographic signature can support verification that an image was authorized by a trusted signing process, while a checksum or hash can help detect accidental corruption or identify an exact artifact. The security value of each mechanism depends on how it is generated, verified, and protected. A signature is only as trustworthy as the signing key and the verification procedure, and a checksum designed solely for accidental-error detection is not a substitute for cryptographic authentication.

Software update mechanisms deserve close scrutiny. An update process should establish which images are acceptable, prevent unauthorized replacement where the system's threat model requires

it, and provide defined behavior if installation is interrupted. It should also address version rollback where reverting to an older release could restore a known vulnerability. Anti-rollback controls must be designed carefully because legitimate recovery may sometimes require installing a previous version. The appropriate policy depends on operational requirements and the consequences of accepting older software.

The firmware package is not necessarily the only component that requires verification. Bootloaders, configuration utilities, companion-computer applications, and supporting services may each have their own update procedures. If one component can alter the behavior or trust decisions of another, the security assessment should include that relationship. An authenticated main firmware image cannot compensate for an unprotected update path that permits a different component to change critical configuration.

Component identification also supports incident investigation. If a vulnerability is discovered in a library, researchers need to determine which firmware releases include the affected version and whether the vulnerable feature is enabled. If two aircraft display different behavior, a comparison of build artifacts and configuration records may reveal a difference that is not apparent from their external appearance.

Configuration and build records should be treated as controlled engineering information. They may reveal operational limits, interface details, software versions, or security settings. Access should be appropriate to the sensitivity of the information, and changes should be recorded so that investigators can determine when and why the system's behavior changed.

# Reviewing Input Validation, Memory Safety, and Error-Handling Mechanisms

Input validation is a fundamental firmware security requirement because embedded systems routinely receive information from sensors, communication interfaces, storage devices, and other software components. Some inputs are generated by trusted internal code, while others originate outside the flight controller's direct control. Even well-intentioned components can produce unexpected data because of hardware faults, software defects, interrupted communication, or incompatible versions.

A robust validation strategy begins by defining what constitutes an acceptable input. The definition may include the permitted data type, length, range, encoding, timing, and relationship to the current system state. For a communication message, validating its structure may require checking that the complete message is present, that its declared length is consistent with the received data, and that mandatory fields are available before the parser accesses them.

Structural validity is only the first requirement. A message can be correctly formatted and still contain an impossible or unsafe value. A sensor measurement may fall outside the physically plausible range, a requested mode may be unsupported on the current aircraft, or a mission parameter may conflict with a configured operating limit. The receiving component should distinguish between data that can be parsed and data that is safe to use.

Validation should occur before data influences a consequential operation. If a received value is copied into a fixed-size buffer, the length must be checked before the copy. If a numeric value is used to calculate an array index, the program must establish that the index is within the valid bounds. If a parameter is used to determine a

memory allocation, the software should prevent arithmetic overflow and unreasonable resource requests.

Memory safety is especially important in firmware written in languages that permit direct memory access and manual management of buffers. A buffer overflow can occur when a program writes beyond the space allocated for an object. An out-of-bounds read can expose unintended data or cause invalid processing. Use-after-free errors arise when software continues to access memory after its allocation has been released. Double-free errors can corrupt memory-management structures or produce unpredictable behavior.

Integer errors can contribute to these defects. A value may overflow its permitted range, become negative when interpreted differently, or be converted to a narrower type. If a length calculation wraps around, the program may allocate less memory than expected and then attempt to write a larger amount of data. Embedded code often uses fixed-width integers to match hardware and protocol requirements, so developers must understand the exact behavior of the types involved.

The use of memory-safe programming techniques can reduce exposure to certain classes of defects. Depending on the project and toolchain, these techniques may include safer abstractions for buffers, explicit bounds checks, ownership rules, static analysis, and restrictions on unsafe operations. Some embedded environments also support hardware protections that help detect or limit memory corruption. These measures are complementary rather than interchangeable, and legacy code may not permit all of them to be adopted at once.

Memory safety has particular importance in software that processes externally supplied data. Communication parsers, file readers, configuration loaders, and update handlers may encounter inputs that differ from normal operational data. Their security should not depend on every upstream component behaving perfectly. A

receiving component should enforce its own requirements before trusting the information.

Concurrency can introduce related problems. If multiple tasks access shared data without suitable synchronization, one task may observe a partially updated structure or a value that changes while it is being processed. In a flight controller, shared data may include sensor readings, operating modes, configuration parameters, and communication state. The resulting defect may appear intermittently and can be difficult to reproduce.

Designers can reduce these risks through clear ownership of shared state, suitable synchronization mechanisms, bounded critical sections, and careful analysis of task interactions. However, synchronization must be compatible with real-time requirements. A mechanism that protects data but blocks a time-critical control task for too long may introduce a different safety problem.

Error handling determines how the firmware responds when a component cannot complete an operation as expected. Some errors are recoverable, such as a temporary communication timeout. Others indicate that the software can no longer trust a critical measurement or configuration value. Treating every error identically is inappropriate because the consequences depend on the affected function and the aircraft's current state.

A useful error-handling design distinguishes between detecting a fault, recording it, containing its effects, and selecting a response. Detection identifies the abnormal condition. Reporting makes the event visible to operators or diagnostic systems. Containment prevents the fault from spreading into unrelated functions. The response establishes what the aircraft should do next, which may involve retrying an operation, rejecting a command, changing modes, restricting functionality, or entering a protective state.

Protective responses must be carefully matched to the aircraft's capabilities. A loss of a noncritical telemetry service may justify recording the problem while continuing normal control. A loss of essential attitude information may require a much more restrictive response. The correct policy depends on the available sensors, redundancy, operating mode, and flight conditions. Security testing should therefore assess whether the response follows the system's documented safety requirements rather than assuming that every fault must trigger an immediate shutdown.

Error messages should be informative without exposing unnecessary sensitive information. Diagnostic records may include component identifiers, fault categories, timestamps, and relevant state information. At the same time, logging should avoid revealing credentials, cryptographic secrets, or other protected material. The design should also limit repetitive logging so that a persistent fault does not exhaust storage or consume excessive processing time.

Silent error recovery can create ambiguity. If a component substitutes a default value whenever a measurement fails, downstream code may continue operating without knowing that the original input was unavailable. In some circumstances, a default value is a valid engineering choice. In others, it can conceal a failure and create an inaccurate estimate of the aircraft's state. The system should make the distinction explicit wherever it affects safe or secure operation.

Watchdog mechanisms can help detect tasks that stop making progress, while health-monitoring services can track missed deadlines, failed sensors, or other abnormal conditions. These mechanisms are not complete solutions by themselves. A watchdog may restart a component without correcting the underlying defect, and repeated restarts may create a cycle of instability. The firmware should define what recovery means, preserve useful diagnostic

evidence, and prevent repeated failures from producing uncontrolled behavior.

A strong review of input validation and error handling follows data from its entry point to its final use. It examines assumptions at each boundary, verifies that invalid values cannot bypass later checks, and assesses whether failures remain contained. It also considers whether the system can report a problem accurately enough for an operator or investigator to understand what occurred.

## Evaluating Software Weaknesses Through Static and Dynamic Analysis

Static and dynamic analysis provide complementary methods for identifying weaknesses in autopilot firmware. Static analysis examines source code or compiled artifacts without requiring the program to execute under the test conditions. Dynamic analysis observes the behavior of a running program. Neither method can establish security by itself, but together they can reveal defects that might remain hidden during ordinary flight testing.

Static analysis includes manual code review, automated source analysis, compiler diagnostics, and inspection of build outputs. These methods can identify suspicious data flows, unsafe memory operations, incorrect bounds checks, unchecked return values, and potentially inconsistent state handling. They are particularly useful when the codebase is large and the reviewer needs a systematic way to prioritize areas for closer examination.

Automated analysis tools apply rules or models to code. Some detect known patterns associated with memory errors or unsafe conversions. Others examine control flow, data flow, resource

management, or compliance with selected coding standards. The quality of the findings depends on the language, the tool's analysis capabilities, and the assumptions used during configuration.

A reported issue must be interpreted in context. Static analysis may identify code that appears unsafe but is protected by a condition the tool cannot fully understand. Conversely, a tool may fail to identify a weakness because the relevant behavior depends on complex interactions between modules or on a configuration that the analysis did not model. Findings should be reviewed, reproduced where practical, and classified according to their demonstrated consequences.

Manual review remains important for logic errors. A function may use memory safely and still grant an inappropriate action, mishandle an operating-mode transition, or apply an incorrect authorization rule. These defects often depend on understanding the intended system behavior rather than recognizing a familiar code pattern. Reviewing requirements alongside implementation helps reveal cases where the code is internally consistent but does not enforce the intended policy.

Dependency analysis complements source review by identifying external components and comparing their versions with available vulnerability information. The result should be checked against the actual build configuration. A vulnerable function that is not compiled or reachable may present a different level of risk from one that is directly exposed through a communication interface. The assessment should document this distinction instead of relying solely on a dependency name or version match.

Dynamic analysis evaluates the compiled program under controlled conditions. For autopilot research, software-in-the-loop simulation provides a practical environment for observing state changes, message handling, sensor processing, and failure responses. A test harness can supply defined inputs and record outputs, while

instrumentation captures relevant internal events. The objective is to compare actual behavior with an expected result derived from documented requirements.

Boundary-value testing is useful when a function accepts numeric ranges, message lengths, or configuration parameters. The researcher examines behavior at the valid limits and just outside them, where appropriate to the interface and test environment. These tests can reveal off-by-one errors, incorrect range assumptions, and inconsistent handling of exceptional values. The test cases should remain controlled and should not depend on uncontrolled access to live aircraft.

Fuzz testing can extend this approach by supplying many automatically generated inputs to a parser or other input-processing component. A fuzzer may vary field lengths, values, message structure, or combinations of fields while monitoring for crashes, assertions, hangs, or other unexpected behavior. In a laboratory, fuzzing is most effective when the target can be reset reliably and when failures can be associated with the input that caused them.

A crash is not the only meaningful finding. The software may accept an invalid command, enter an inconsistent state, stop processing telemetry, or consume excessive resources without terminating. Instrumentation should therefore capture more than process exit status where practical. Relevant observations may include error reports, task timing, state transitions, resource use, and the system's ability to recover.

Dynamic analysis should account for the difference between a test harness and the full application. A parser tested in isolation may behave differently when integrated with the scheduler, communication layer, and state-management logic. Conversely, a full-system test may make a defect difficult to localize. A productive approach uses focused component tests to identify specific

weaknesses, followed by integration tests to determine whether the behavior has a meaningful effect on the complete system.

Sanitizers and runtime instrumentation can improve defect detection when the target architecture and build environment support them. These tools may detect invalid memory access, certain forms of undefined behavior, or other programming errors during execution. Their applicability varies across embedded platforms, and instrumentation can change timing or memory usage. A result obtained from an instrumented build should therefore be interpreted with awareness of those differences.

Code coverage can indicate which parts of the software were executed during a test suite. Statement coverage records whether individual statements ran, while branch coverage examines whether alternative decision outcomes were exercised. Higher coverage does not prove the absence of defects, but coverage information can identify important code that has not been exercised and help researchers improve their test design.

Security testing should also examine negative behavior. When an input is invalid, the expected result may be rejection without changing system state. When communication is lost, the expected result may be a documented protective response. When a firmware image fails verification, the expected result may be refusal to execute it. Tests should verify both the rejection and the absence of unintended side effects.

Findings should be evaluated according to their technical evidence and operational significance. A defect that causes a diagnostic application to display an incorrect label differs from one that permits unauthorized modification of a safety-critical configuration. Severity depends on factors such as reachability, required privileges, reliability of reproduction, potential consequences, and the effectiveness of existing safeguards. The assessment should

distinguish a confirmed weakness from a theoretical concern whose necessary conditions have not been established.

Every confirmed finding should have a reproducible record. This record should identify the affected software revision, build configuration, test environment, input or triggering condition, observed behavior, expected behavior, and relevant logs. Where a defect has been corrected, the same test should be repeated against the modified build to determine whether the original behavior has been resolved. Regression tests should then preserve the test case so that future changes do not silently reintroduce the problem.

Static and dynamic analysis are most valuable when their findings inform one another. Static analysis may identify a suspicious length calculation, while dynamic testing determines whether a relevant input can reach that calculation and produce an observable failure. A dynamic test may reveal an unexpected state transition, prompting a source review that identifies the condition responsible. This iterative process connects code-level evidence with actual software behavior.

The final assessment should state what was tested and what remains outside the test's scope. A clean static-analysis report does not prove that the firmware is secure, and a successful simulation does not establish that the same behavior will occur on every physical aircraft. Processor-specific timing, hardware faults, peripheral behavior, deployment configuration, and external dependencies may require separate evaluation. Careful reporting preserves these distinctions and gives engineering teams a sound basis for deciding which weaknesses require correction, further testing, or additional safeguards.

# Firmware Analysis and Reverse Engineering

## Understanding Binary Formats, Memory Maps, and Executable Structures

Firmware analysis and reverse engineering provide a systematic way to understand how embedded software is organized, how it interacts with hardware, and how its behavior can be evaluated when complete source code is unavailable. In unmanned aircraft systems, these techniques are valuable for assessing software integrity, identifying undocumented components, verifying update procedures, investigating unexpected behavior, and understanding the relationship between a compiled firmware image and the functions performed by an autopilot.

Firmware analysis begins with a fundamental distinction between a program's source code and its executable representation. Source code expresses instructions in a programming language that developers can read and maintain. A compiler translates those instructions into machine code suited to a particular processor architecture, while a linker combines compiled modules and resolves references between them. The resulting executable may then be packaged, signed, compressed, or otherwise transformed before it is installed on a flight controller.

The transformation process introduces information that is not always obvious from the final file. Function names may be removed, debugging symbols may be excluded, and separate software modules

may be combined into a single image. Hardware-specific instructions and memory addresses may also differ between builds of the same software. Consequently, a firmware image cannot be understood reliably by examining its filename or extension alone.

A file described as firmware might contain a complete executable, a bootloader, a collection of images for different hardware components, or a container that includes metadata and compressed payloads. Some update packages contain only the changes required to move from one version to another. Others contain a complete replacement image. The structure depends on the development and distribution process, so analysis should begin by identifying the file's actual format and intended role.

Binary formats provide rules for interpreting executable content. A format may describe the processor architecture, the locations of executable sections, the organization of initialized data, and the entry point at which execution begins. Some executable formats preserve symbolic information that helps an analyst associate machine instructions with functions or variables. Other formats are designed primarily for compact deployment and may omit information useful during debugging.

Embedded systems commonly use executable formats suited to their toolchains and operating environments. An executable file used during development may contain a rich set of section headers and debugging information, while the firmware installed on a microcontroller may be a flat binary with little structural metadata. The absence of an elaborate executable header does not mean that the image lacks organization. Its placement in memory may be determined by the processor's startup conventions, the board's linker configuration, and the firmware packaging process.

A sound examination distinguishes between the file's logical structure and the way the target hardware uses it. The file may contain code, constants, initialized data, configuration defaults, and

other resources. Some of these elements are copied into RAM during startup, while others remain in nonvolatile memory. Uninitialized variables may occupy memory without corresponding bytes in the stored image because their initial values are established by startup code.

This distinction is important when comparing the file size with the memory requirements of a running application. The executable's size does not necessarily represent its total runtime footprint. A program may occupy a relatively small amount of flash memory while requiring substantially more RAM for stacks, buffers, sensor data, task state, and dynamically allocated objects. Conversely, a large firmware package may contain several images or resources that are not loaded into memory simultaneously.

Memory maps describe how different regions of a processor's address space are assigned to hardware and software. Depending on the platform, these regions may include flash memory, static RAM, peripheral registers, bootloader storage, persistent configuration areas, and memory reserved for specialized functions. The map helps explain where executable code is expected to reside, where writable data is placed, and which addresses correspond to hardware interfaces rather than ordinary memory.

The linker configuration is often central to this analysis. It specifies how compiled sections are arranged and may reserve particular regions for startup code, application code, persistent data, or update metadata. A firmware image built for the wrong memory layout may fail during startup, overwrite an unintended region, or interact incorrectly with hardware. Even when the image is syntactically valid, its suitability depends on the processor and board for which it was produced.

The processor's execution model also influences interpretation. Some architectures execute instructions directly from nonvolatile memory, while others copy portions of the program into RAM before

execution. Certain processors provide distinct address spaces or special instructions for accessing peripheral registers. Analysts must understand these characteristics before treating an address as an ordinary memory location or interpreting a sequence of bytes as machine instructions.

Executable structures help establish the relationship between these regions. Code sections typically contain instructions, read-only sections may contain constants and lookup tables, and writable sections hold data that changes during execution. Startup routines commonly initialize the runtime environment, copy initialized data to its execution location, clear designated memory regions, and transfer control to the main application logic.

The entry point listed in an executable may not correspond to the first meaningful operation performed after power is applied. Hardware reset behavior, processor-specific startup code, and a bootloader may execute before the principal application. This is why understanding the complete startup path is essential when investigating boot behavior or evaluating whether a firmware image follows the expected initialization sequence.

Reverse engineering reconstructs aspects of a program's design from its executable representation. The process may involve disassembly, which converts machine instructions into assembly-language representations, and decompilation, which attempts to produce a higher-level approximation of the original program. These methods can help identify control flow, function boundaries, data references, and interactions with hardware.

The output of a decompiler is an interpretation rather than a recovered copy of the original source code. Variable names, comments, and much of the original type information may be absent. Compiler optimization can also transform the original structure, combine functions, eliminate redundant operations, or rearrange instructions. A responsible analyst treats reconstructed code as a

hypothesis that must be checked against the executable's behavior and surrounding evidence.

Architecture identification is an early requirement because machine instructions have meaning only in relation to the correct instruction set and execution mode. An incorrect architecture setting can produce misleading disassembly, false function boundaries, and incorrect conclusions about control flow. File metadata, known build information, processor documentation, and the structure of the image can help establish the correct interpretation.

Analysis becomes more reliable when the researcher records the assumptions used to interpret the binary. These assumptions may include the target processor, the likely memory layout, the expected startup address, and the relationship between image sections and hardware regions. When new evidence contradicts an assumption, the interpretation should be revised rather than forcing the evidence to fit the original theory.

# Examining Firmware Images and Legitimate Update Packages

Firmware images are the principal artifacts examined during many embedded security assessments. They represent a particular software build and can reveal how the system is packaged for installation, which components are included, and what mechanisms are intended to protect software delivery. Examining these artifacts is especially useful when performed against firmware obtained from an authorized development environment or an official distribution channel.

The first step is to establish provenance. An analyst should record where the image came from, when it was obtained, which product or board it is intended to support, and what release information accompanies it. If the image was supplied as part of an authorized update package, the original package should be preserved alongside the extracted payloads. This maintains the relationship between the distributed artifact and the files used for analysis.

File extensions and filenames provide clues, but they are not definitive evidence of format. A file may use a generic extension, include several embedded payloads, or contain a wrapper around the actual firmware. Examination should therefore establish whether the artifact is a standalone executable, a compressed image, an archive, a signed container, or a package containing several related components.

A firmware update package may include a manifest that describes the target device, version, image size, supported hardware revision, or other installation requirements. The manifest may also identify cryptographic hashes or signatures associated with the payload. Such information can help establish what the update process is intended to verify before installation.

The analyst should distinguish the package format from the firmware format. The outer package may be designed for transport and verification, while the payload follows the memory layout expected by the target processor. A package may also include separate images for the bootloader, application processor, radio module, or companion computing system. Treating the entire package as a single executable can obscure these distinctions.

Preserving the original artifact is an essential discipline. Analysis tools may extract, transform, or normalize files, and some workflows may modify data unintentionally. The original should remain unchanged while examination proceeds on working copies. A

cryptographic hash recorded at acquisition provides a practical means of checking whether the file has changed during handling.

The next task is to determine how the image is organized. Recognizable headers, section metadata, file signatures, and documented package structures can help establish its composition. Where the image lacks explicit structural information, the analyst may need to compare its size and layout with known build outputs or inspect it using tools suited to the identified architecture. Any inferred boundaries should be recorded as interpretations rather than presented as confirmed facts.

Compression and encryption require different treatment. Compression changes the representation of the data to reduce its size but is generally intended to be reversible without a secret key. Encryption protects the contents from being read without the required cryptographic material. A file that appears to contain random-looking data is not necessarily encrypted; it may also be compressed, signed, or structured in a way that is not immediately recognizable.

If an official update package uses encryption, the assessment should respect the intended access controls and remain within the authorization provided for the work. Security review does not require bypassing protections indiscriminately. Often, the update manifest, installation documentation, verification behavior, or authorized development build provides sufficient evidence to evaluate the design without attempting to defeat confidentiality mechanisms.

Comparing firmware versions can reveal how the software changes over time. A comparison may identify differences in executable sections, metadata, configuration defaults, or included resources. However, binary differences do not automatically indicate functional changes. Compiler versions, build timestamps, section ordering, and other build characteristics can change the output even when the application behavior remains substantially the same.

A meaningful comparison therefore considers the build process and the intended changes. If source code and build records are available, the analyst can compare the relevant revisions and examine whether the observed binary differences correspond to the documented changes. When source code is unavailable, repeated patterns and structural differences can guide further investigation, but conclusions should remain proportionate to the available evidence.

Firmware packages also provide an opportunity to examine update compatibility rules. A well-designed installer should determine whether an image is intended for the target hardware and whether its version and format meet installation requirements. The system should not assume that any syntactically valid image is suitable for any device. Incorrect target selection can create operational failures even when the software itself is authentic.

Interrupted updates are another important consideration. An update may fail because of power loss, a damaged transfer, insufficient storage, or an unexpected reset. The design should specify whether the system can recover, retain a known working image, or enter a documented recovery mode. An analysis of package structure can help identify whether separate images or metadata appear to support such a process, although the package alone may not establish how the target device implements recovery.

A firmware image is evidence of a particular build, not a complete description of the deployed system. The actual behavior may also depend on persistent configuration, hardware revisions, bootloader state, external components, and installation history. A responsible examination documents these limits and avoids treating one extracted file as proof of the security properties of every device using the same product name.

# Identifying Embedded Resources, Libraries, and Configuration Parameters

Firmware often contains more than executable instructions. It may incorporate text strings, error messages, protocol identifiers, default parameters, calibration constants, mathematical tables, cryptographic material, and resources used by diagnostic or maintenance functions. Identifying these elements can help researchers understand the software's capabilities and locate components that deserve closer review.

Readable strings are a useful starting point because they can reveal descriptive information that remains present in the compiled image. Examples include diagnostic messages, version identifiers, module names, protocol labels, or references to supported hardware. These strings may suggest which functions exist and how the software communicates errors to operators. Their absence, however, does not prove that a function is absent, because compilers and build processes may remove or transform descriptive information.

Strings should be interpreted in context. A message may exist in a shared library that is included in the build but never used by the selected firmware configuration. A label may be retained for compatibility with several hardware targets. Conversely, a firmware function may be active without containing any obvious descriptive text. A string is a clue that directs further examination, not proof of a particular runtime capability.

Embedded libraries can sometimes be identified through version information, copyright notices, recognizable metadata, symbol names, or characteristic code structures. These clues help establish which components may be included in the image. The identification

can then be compared with build manifests, source repositories, or documented dependency information where available.

Library identification requires caution because compiler optimization and local modifications can alter the appearance of compiled code. Two versions of a library may produce similar machine instructions, while the same source may produce noticeably different binaries under different compiler settings. A confident identification should rely on multiple supporting indicators rather than a single matching sequence of bytes.

The presence of a library also does not prove that its functionality is reachable. Some build systems link code that is not called by the active application, while others remove unused functions during optimization. Security assessment should establish whether the relevant component is included in the deployed image and whether the affected code can be reached through a realistic execution path.

Configuration parameters are particularly important because they influence behavior without necessarily changing the executable instructions. Default values may be compiled into the firmware, while user-selected settings may be stored in nonvolatile memory. Other values may be generated during calibration or established when the aircraft starts.

The meaning of a parameter may not be obvious from its numeric representation. A stored value could represent a timeout, a mode selection, a scaling factor, or a flag controlling whether a feature is enabled. Interpretation should be grounded in documentation, source definitions, known configuration formats, or controlled observations. Guessing from a value alone can lead to incorrect conclusions.

Configuration structures may also contain version fields or identifiers used to determine how data should be interpreted. When

the software evolves, a parameter may be renamed, assigned a different range, or given a revised meaning. If the firmware reads an older configuration without appropriate migration or validation, it may interpret the stored data incorrectly. Such problems are especially relevant when settings persist across multiple firmware updates.

A useful examination distinguishes between immutable defaults and mutable operational settings. Immutable defaults may be embedded in read-only sections of the image, while mutable settings may reside in separate storage. The actual arrangement depends on the platform. The analyst should determine whether the values are loaded at startup, modified during operation, or written back to persistent storage.

Sensitive material requires special care. Firmware may contain cryptographic constants, certificates, public keys, or other data used in authentication and communication. Not all such material is secret. A public verification key, for example, may be expected to reside in the firmware so that the device can validate signed updates. A private signing key embedded in a widely distributed image would present a very different security concern.

The distinction between public and secret cryptographic material is fundamental. Public keys and certificates can support verification without compromising the corresponding private key. Shared secrets and private keys, by contrast, require protection appropriate to their role. Their security should be assessed in relation to how they are generated, stored, provisioned, and used, rather than simply whether they appear in a binary.

Configuration and resource analysis can also expose operational assumptions. A compiled default may permit a feature that should be disabled in a particular deployment, or a diagnostic interface may remain available when it is not needed. These observations should be assessed against the intended product requirements and

documented deployment policy. A feature's presence is not automatically a vulnerability, but an unnecessary feature can increase the amount of code and functionality that must be secured.

Resource identification is most useful when it leads to a clear technical question. If a binary contains a version string for a communication library, the analyst can determine whether the library is included and whether its relevant functions are reachable. If a configuration value appears to control a safety mechanism, the analyst can investigate how the value is validated and applied. This process turns observations into testable findings rather than a collection of unexplained binary details.

## Assessing Secure Boot, Firmware Signatures, and Update Integrity

Firmware authenticity is essential when an embedded system must distinguish approved software from unauthorized or corrupted images. In an autopilot, the software may determine how sensor information is interpreted, how control commands are generated, and how the aircraft responds to faults. If an unauthorized image can be installed or executed, other security controls may be weakened or bypassed. Secure boot and authenticated update mechanisms are intended to reduce this risk, although their effectiveness depends on the complete implementation.

Secure boot establishes a chain of verification from an initial trusted component to the software that is ultimately executed. Depending on the hardware, the first stage may be enforced by processor features, immutable boot code, or another protected mechanism. That stage verifies a later component, which may then verify the main firmware or additional software modules. Each stage should have a defined trust relationship with the next.

The chain is only as strong as its earliest trusted element and the integrity of the verification process. If the first component can be replaced without detection, later checks may no longer provide the intended assurance. Similarly, if the verification policy can be altered by unauthorized software, an attacker may be able to weaken the protection without changing the image that is ultimately loaded.

Firmware signatures commonly use public-key cryptography. A trusted private key signs a defined representation of the image or its metadata, and the receiving device uses the corresponding public key to verify the signature. If the signature verification succeeds under the expected policy, the device has evidence that the signed data was approved by the holder of the signing key and has not been modified in a way that invalidates the signature.

A valid signature does not establish that the firmware is free from defects. It establishes a relationship between the image, the signature, and the signing key. An authorized image can still contain programming errors, outdated dependencies, unsafe defaults, or vulnerabilities. Signature verification should therefore be understood as a measure of authenticity and integrity, not a substitute for software assurance.

The signed data must also be defined carefully. If a signature covers only the executable payload but excludes security-relevant metadata, an attacker may be able to alter an excluded field in ways that affect installation or execution. A sound design establishes which components are covered by verification and ensures that security-sensitive metadata cannot be changed independently without detection.

Hash functions often contribute to this process by producing a fixed-length digest of the data being protected. A cryptographic hash can help identify changes to an image, but it does not authenticate the

origin of the image by itself. If an attacker can replace both the image and an unprotected hash value, the comparison may still succeed. Authentication requires a trust mechanism, such as a digital signature or an appropriately protected message authentication code, depending on the design.

Update integrity includes more than verifying a signature at the beginning of installation. The update process should preserve the relationship between the verified image and the bytes that are actually written and executed. It should also handle interrupted transfers, storage errors, incompatible targets, and incomplete installations. Otherwise, a system may verify one artifact but install a different or incomplete representation of it.

Version management introduces another security concern. An older firmware release may have a valid signature while containing a vulnerability that has since been corrected. If the update policy accepts every historically signed release without additional restrictions, an authorized but outdated image may restore known weaknesses. Anti-rollback controls can restrict installation of older versions, but they must be balanced against legitimate recovery needs and the possibility that an update itself introduces a serious defect.

Key management is central to the trust model. Signing keys must be protected against unauthorized use, and the system must define how verification keys are installed, replaced, or revoked. If a signing key is compromised, the consequences may extend to every device that trusts it. A mature update architecture therefore considers key storage, access control, backup procedures, key rotation, and the process for responding to compromise.

Different components may have separate trust requirements. A flight controller, bootloader, communication module, and companion computer may use distinct update mechanisms. Protecting only the main application image leaves potential gaps if another component

can alter the boot process, supply untrusted configuration, or influence security-critical behavior. The assessment should identify which components participate in the chain of trust and how their verification procedures relate to one another.

Firmware analysis can evaluate these protections without assuming that every system supports the same security features. Documentation may describe the expected verification policy, while authorized test images and development tools can demonstrate how the update process handles approved and rejected artifacts. A controlled assessment can establish whether integrity checks occur, what information is verified, and how the system reports a failed verification attempt.

The analyst should also distinguish between a cryptographic mechanism being present and its correct enforcement. A package may contain a signature, yet the installer might not verify it under every installation path. A device may support secure boot but permit a maintenance mode with different verification requirements. These possibilities should be investigated through documentation and authorized testing, with conclusions limited to the tested configuration.

Recovery behavior deserves particular attention. A system that rejects an invalid image must still have a safe method of recovering from a failed legitimate update. Recovery procedures should not silently disable essential verification protections or introduce an unrestricted path for installing untrusted software. The design must reconcile availability with authenticity, especially when the device is difficult to access physically.

The outcome of this assessment should establish what the available evidence demonstrates about image integrity, signing policy, trust anchors, version handling, and recovery. If certain details cannot be confirmed from the authorized artifacts, they should be recorded as

unresolved rather than inferred from the presence of cryptographic metadata alone.

## Documenting Findings and Reproducing Results in an Authorized Laboratory

Firmware reverse engineering is most valuable when the resulting observations can be independently reviewed and reproduced. A technically interesting interpretation of a binary is not sufficient on its own. Other researchers and engineers need to understand which artifact was examined, what methods were used, which assumptions were made, and how the conclusions follow from the evidence.

The laboratory record should begin with the identity of the artifact. This includes the filename, acquisition source, collection date, intended hardware target, declared software version, and cryptographic hash. The original file should be preserved unchanged, while extracted or transformed files should be stored as separate working artifacts. This makes it possible to distinguish the original evidence from the products of analysis.

The analysis environment should also be documented. Relevant information may include the operating system, tool versions, processor architecture settings, analysis configuration, and any scripts used to process the firmware. When a result depends on a particular tool option or interpretation setting, recording that choice helps another analyst reproduce the same observation.

Each significant conclusion should be connected to its supporting evidence. If the analysis identifies an executable region, the record should explain how that region was recognized and what information supports the proposed interpretation. If a component is believed to

be a particular library, the report should describe the indicators supporting that identification and acknowledge any uncertainty. This approach prevents a tentative observation from becoming an unsupported statement of fact.

Reverse engineering often involves competing hypotheses. A region of bytes might represent compressed data, a lookup table, an executable section, or a proprietary structure. Rather than selecting one explanation immediately, the analyst should compare the possibilities against known file boundaries, processor conventions, surrounding data, and documented package formats. Confidence increases when independent observations support the same interpretation.

Reproduction requires more than preserving screenshots or copying a few output lines. The laboratory should retain the relevant input files, analysis settings, commands or procedures, and generated results needed to repeat the examination. Where a proprietary or unavailable tool limits reproducibility, that limitation should be stated. The objective is to provide enough information for another qualified researcher to evaluate the result without requiring access to undocumented assumptions.

When firmware behavior is examined dynamically, the record should include the test hardware or simulator configuration, the software build, the initial state, the procedure followed, and the observable outcome. If an update package is tested, the report should identify the test device or authorized environment and explain how acceptance or rejection was established. Tests should remain within the authorized scope and should avoid modifying operational aircraft or unrelated systems.

Findings should distinguish between confirmed defects, security concerns requiring further investigation, and informational observations. A confirmed defect has supporting evidence that the implementation behaves contrary to an established requirement or

permits an unintended condition. A concern may be plausible but lack enough evidence to establish reachability or impact. An informational observation may describe a design feature without demonstrating that it creates a weakness.

The assessment should describe impact carefully. Discovering an undocumented parameter does not establish that it can be modified by an unauthorized party. Identifying a cryptographic library does not prove that the system uses it incorrectly. Finding a signature in an update package does not prove that the target device enforces signature verification. Each claim requires evidence appropriate to the specific conclusion.

A strong technical report explains the conditions under which the finding applies. The affected firmware version, hardware target, configuration, and relevant dependencies should be identified where known. If the behavior differs between builds, that difference should be documented. Such precision prevents a finding from being generalized to devices or versions that were not examined.

Reproducibility also supports remediation. When a defect is corrected, the original test can be repeated against the revised firmware to determine whether the expected behavior has been restored. The same test should be retained as a regression test where practical. This gives developers a way to check whether future changes reintroduce the problem and whether the correction has unintended effects on related functions.

Responsible handling of findings is part of professional firmware analysis. Research should be conducted on software and devices for which the analyst has appropriate authorization. Sensitive artifacts, private keys, credentials, and information that could expose operational systems should be protected. If a finding affects a deployed product, the appropriate disclosure process should provide the responsible maintainer with enough technical information to investigate and correct the issue.

A disciplined record preserves the difference between what the analyst observed, what the evidence supports, and what remains uncertain. That distinction is essential in reverse engineering because the available information is often incomplete. A well-supported analysis does not require every question to have an immediate answer; it requires that the boundaries of current knowledge be stated clearly and that the reasoning behind each finding remain open to verification.

# Radio-Frequency Communications and Wireless Security

## The Fundamentals of Radio Links Used by Unmanned Aircraft

Radio-frequency communication is a fundamental part of many unmanned aircraft systems because it allows information to move between an aircraft and the equipment or services supporting its operation. Through wireless links, an operator may issue commands, receive telemetry, monitor system health, transmit mission information, or exchange data with other authorized components. These functions make radio communication essential to aircraft operation, but they also introduce security and reliability concerns that differ from those associated with a purely wired system.

A radio link transfers information through electromagnetic energy rather than through a physical cable. A transmitting device converts information into a signal suitable for transmission, an antenna radiates electromagnetic energy, and a receiving antenna captures part of that energy. The receiver then processes the signal to recover the transmitted information. The practical quality of this process depends on the radio hardware, antenna characteristics, propagation environment, transmission parameters, receiver sensitivity, and the design of the communication protocol.

An unmanned aircraft may use several wireless connections rather than a single universal link. A dedicated control link can connect the operator's controller to the aircraft, while a telemetry channel carries information about position, battery condition, operating mode, and other system measurements. A separate wireless connection may support a payload, a companion computer, or an onboard network. Some aircraft also use cellular connectivity or satellite communications for remote monitoring and operations beyond the range of a conventional local radio link.

These connections should not be treated as interchangeable. They may operate on different frequencies, use different modulation methods, provide different data rates, and enforce different security requirements. A high-bandwidth payload connection may be suitable for transmitting video but poorly suited to time-critical control messages. A low-bandwidth telemetry channel may provide reliable status updates without having the capacity to transport large data files. Security analysis must account for these functional differences.

The design of a radio link begins with the requirements of the application. Control communication places particular importance on predictable delivery, manageable latency, and clear handling of lost or delayed messages. Telemetry communication must convey useful information without overwhelming the available bandwidth. Video and sensor-data links may require substantially greater throughput, while remote management channels may place greater emphasis on authentication, confidentiality, and continuity of service.

Radio communication also operates under physical constraints. Electromagnetic signals weaken as they travel, and the amount of energy received depends on distance, antenna orientation, obstructions, reflections, and environmental conditions. Buildings, terrain, vegetation, aircraft structures, and other objects can absorb, reflect, or scatter radio energy. A link that performs reliably during

an open-field test may behave differently near buildings or in a crowded operating environment.

The relationship between transmitter power, antenna gain, propagation loss, and receiver sensitivity is often described through a link budget. A link budget estimates whether enough signal energy will reach the receiver for the intended communication method to function. It accounts for the transmitter's output, gains introduced by antennas, losses along the transmission path, and other relevant factors. The result helps engineers assess whether a connection has an adequate margin under expected conditions.

Link margin is particularly important because real operating environments rarely remain constant. Aircraft movement changes antenna orientation, and the signal may pass through different propagation conditions as the vehicle changes position. A link that operates only slightly above the receiver's minimum requirements may become unreliable when the aircraft banks, an obstruction enters the path, or interference increases.

Antenna placement can be as important as transmitter power. The antenna's radiation pattern determines how energy is distributed in different directions. Some antennas provide broad coverage, while others concentrate energy more strongly in particular directions. Their orientation and proximity to conductive materials can influence performance. On an aircraft, motors, wiring, batteries, payloads, and the airframe itself may affect antenna behavior or introduce unwanted electrical noise.

The frequency of operation influences propagation characteristics and antenna design. Lower frequencies often experience different diffraction and penetration behavior than higher frequencies, while higher frequencies can support compact antennas and wider bandwidths under suitable conditions. These are general tendencies rather than guarantees. Actual performance depends on frequency,

antenna characteristics, the environment, and the specific radio technology.

The physical medium creates an important security distinction. A wired connection generally requires access to the cable or an attached interface, whereas a radio signal may be detectable beyond the immediate operating area. This does not mean that every wireless transmission can be decoded or successfully received by an outside observer. Signal strength, distance, interference, antenna characteristics, modulation, and cryptographic protection all influence what an observer can learn or accomplish.

Nevertheless, wireless systems should be designed with the expectation that radio transmissions may be observable by parties who are not authorized to use them. Confidential information should not rely on the secrecy of the transmission frequency or the assumption that an outsider cannot identify the protocol. Authentication, encryption, appropriate access controls, and robust message validation are therefore central to wireless security.

Radio links also require a clear understanding of availability. A receiver may fail to recover information even when the sender is legitimate and the data is correctly formatted. Interference, weak signal conditions, receiver overload, damaged antennas, and hardware faults can all disrupt communication. Security mechanisms cannot eliminate every physical-layer problem, so the aircraft must have a defined response when a link becomes unreliable.

The response should be appropriate to the function of the lost connection. Losing a video feed may not require the same reaction as losing a control link. Losing telemetry may leave the aircraft controllable while reducing the operator's understanding of its condition. The system should distinguish between these cases rather than treating every communication failure as identical.

A sound radio architecture defines the purpose of each connection, the information it carries, the authority it provides, and the expected behavior when communication becomes unavailable. This creates a foundation for assessing wireless security without confusing normal radio limitations with exploitable software weaknesses.

## Control Channels, Telemetry Links, and Data Transmission

Control channels provide the communication path through which an operator or authorized supervisory system requests changes to an aircraft's behavior. Depending on the aircraft design, the messages may represent pilot inputs, operating-mode selections, mission commands, or other permitted actions. The control link is therefore security-sensitive because it may carry instructions that influence movement, navigation, or the activation of particular functions.

A control channel is not necessarily a continuous stream of direct motor commands. In many modern systems, the flight controller performs the immediate stabilization work onboard, while the operator provides higher-level requests. The aircraft's autopilot interprets these requests in relation to its current state, configured limits, and operating mode. This separation allows the aircraft to maintain local control behavior even when the external command stream is relatively low-bandwidth.

The distinction between requested behavior and actual control authority is important. A message may request a mode change, but the autopilot should determine whether the transition is permitted under current conditions. A command that is valid in one state may be inappropriate in another. The communication interface should therefore validate the sender and message structure, while the flight-

control system enforces the rules governing whether the requested action is acceptable.

Telemetry links serve a different but closely related purpose. They provide information about the aircraft's estimated position, orientation, battery state, navigation status, operating mode, and system health. Depending on the design, telemetry may also include diagnostic messages, mission progress, sensor measurements, and information about communication quality.

Telemetry is often essential to safe operation because the operator cannot directly observe every internal state of an aircraft. However, telemetry values are measurements or estimates rather than infallible representations of reality. A position estimate may become less reliable when navigation signals degrade, and a battery estimate may depend on calibration and the assumptions of the monitoring system. Security assessment should therefore consider the source, freshness, and integrity of the data rather than treating every displayed value as authoritative.

The distinction between telemetry reception and command authority is especially important. An application may need permission to observe an aircraft without requiring permission to alter its configuration or request flight actions. Separating these privileges reduces the consequences of a compromised monitoring application or an unauthorized client that gains access to the communication network.

Some communication systems use a shared radio channel for both commands and telemetry. Others separate the functions across different channels, frequencies, or network connections. A shared link can simplify hardware and reduce system complexity, but it also requires appropriate management of competing traffic. Telemetry bursts or large payload transfers should not prevent time-sensitive control messages from receiving the service they require.

Quality-of-service mechanisms, message prioritization, and traffic management can help preserve this separation. Their effectiveness depends on the implementation. A priority label alone does not guarantee timely delivery if a component can monopolize processing resources or saturate the radio channel. The complete system must account for both network capacity and the behavior of the software that generates and processes messages.

Data transmission may involve more than control and telemetry. A camera, mapping sensor, environmental instrument, or other payload may produce substantial quantities of information. A companion computer may process that information onboard and transmit selected results to a ground station. These data paths can create additional security boundaries because the payload, companion computer, and flight controller may have different permissions and levels of trust.

A payload connection should not automatically inherit authority over the flight-control system. If a camera or data-processing application is compromised, the consequences should be limited by the interfaces it is permitted to use. A design that separates payload data from flight-critical commands can reduce the chance that a fault in one subsystem affects the entire aircraft.

Communication protocols define how information is represented and interpreted. They may specify message identifiers, field types, lengths, sequence information, source identifiers, and error-detection mechanisms. Some protocols also support authentication or encryption, while others rely on protection provided by a separate transport layer. Understanding where each protection is applied is necessary to assess the actual security of the link.

Error-detection mechanisms, such as checksums or cyclic redundancy checks, help identify accidental corruption during transmission. They do not necessarily prevent intentional

modification. An attacker who understands the format may be able to generate a new message with a valid error-detection value. Cryptographic message authentication provides a different type of protection by allowing a receiver to verify that a message was produced by an entity possessing the appropriate secret key.

Sequence numbers and timestamps can help receivers identify duplicate, delayed, or out-of-order messages. They are useful for preserving communication consistency, but their security value depends on how they are validated. If the receiver accepts old messages without checking their freshness, a previously valid instruction may be processed under conditions different from those for which it was originally intended. Authentication and freshness checks should therefore be considered together.

Wireless protocols also need clear rules for retransmission and loss. Repeating a message can improve delivery under some conditions, but repeated commands may have different consequences depending on whether the operation is idempotent. An idempotent operation produces the same intended result when repeated, while a non-idempotent operation may cause additional effects each time it is processed. Designers should consider this distinction when defining command semantics and recovery behavior.

When a control link becomes unavailable, the aircraft should follow its documented failsafe policy. Depending on the aircraft and mission, that policy may involve maintaining a safe operating mode, attempting to regain communication, returning to a defined location, landing, or performing another predetermined action. The appropriate response depends on the available navigation capabilities, energy reserves, operating environment, and safety requirements.

A communication failure should also be reported clearly to the operator. The system should distinguish between loss of telemetry, loss of command capability, and degradation of a payload connection

wherever its architecture permits. This improves situational awareness and helps prevent the operator from assuming that all communication functions remain available when only part of the link is working.

## Frequency Bands, Modulation, Interference, and Signal Quality

Wireless communication depends on the selection and use of radio-frequency spectrum. Different frequency bands have different propagation characteristics, regulatory requirements, antenna dimensions, and available bandwidths. The appropriate choice depends on the communication purpose, the intended operating environment, the required data rate, and the regulations governing the location of operation.

Many unmanned aircraft systems use frequencies within bands allocated for particular categories of radio communication. Some devices operate in licence-exempt bands subject to technical restrictions, while other systems use licensed spectrum or service-specific allocations. The fact that a band is licence-exempt does not mean that it is unrestricted. Limits may apply to transmitter power, channel use, antenna configuration, duty cycle, or interference with other users.

Frequency selection should be treated as a regulatory and engineering decision. A radio that functions correctly in one country may not be permitted to operate with the same settings elsewhere. Operators and researchers must account for the applicable rules rather than assuming that a commercially available device can be used at any power level or on any channel.

Modulation describes how information is represented in a radio signal. A transmitter varies one or more properties of a carrier wave according to the information being sent. Common approaches include changing amplitude, frequency, or phase, as well as more complex schemes that combine several dimensions. Digital communication systems often use modulation methods designed to balance throughput, robustness, implementation complexity, and spectral efficiency.

More complex modulation can transmit more information within a given period or bandwidth, but it may require a stronger or cleaner received signal to maintain reliable decoding. A system that selects a more robust modulation mode may tolerate weaker signal conditions at the cost of lower data throughput. Adaptive systems can change modulation and coding parameters as conditions vary, provided that the transmitter and receiver coordinate those changes correctly.

Coding adds structured redundancy to transmitted data so that the receiver can detect or correct some errors. Forward error correction can improve reliability without requiring every corrupted transmission to be repeated. Interleaving may distribute the effects of short bursts of interference across a larger sequence of data, making recovery more effective for some coding schemes. These mechanisms improve resilience but cannot guarantee successful communication under every condition.

Signal strength is an important measurement, but it does not fully describe link quality. A receiver may detect a strong signal and still have difficulty decoding the intended data if interference is substantial or the signal is distorted. Conversely, a relatively weak signal may remain usable when the noise level is low and the receiver is sufficiently sensitive.

Signal-to-noise ratio compares the strength of the desired signal with the level of background noise. A higher ratio generally supports more

reliable reception, although the required value depends on the modulation, coding, receiver design, and target error rate. Signal-to-interference-plus-noise ratio also accounts for interfering transmissions and can provide a more realistic view of performance in a busy radio environment.

Interference can originate from unrelated wireless devices, nearby transmitters, electrical equipment, poorly designed electronics, or other sources of electromagnetic energy. Co-channel interference occurs when transmissions overlap on the same channel, while adjacent-channel interference can arise when signals on nearby frequencies affect the receiver. Harmonics and unwanted emissions may also create problems when equipment does not meet its intended spectral characteristics.

The practical effect of interference depends on its location, duration, power, frequency, and relationship to the receiver's processing. A brief disturbance may cause isolated packet loss, while sustained interference may reduce throughput or interrupt communication altogether. Interference can also affect different channels unevenly, producing a situation in which one service remains available while another becomes unreliable.

Multipath propagation is another important factor. Radio waves may reach a receiver by several paths after reflecting from buildings, terrain, or other surfaces. The received signals may reinforce or partially cancel one another depending on their relative phase and timing. As the aircraft or surrounding objects move, these relationships can change, producing fluctuations in received signal quality even when the transmitter's power remains constant.

Fading describes variations in received signal strength caused by propagation effects. Some systems use antenna diversity, multiple-input multiple-output techniques, frequency selection, or adaptive coding to reduce the impact of fading. The suitability of each method depends on the radio design and the constraints of the aircraft.

Antenna orientation and polarization also influence reception. If the transmitting and receiving antennas have mismatched polarization, some signal energy may be lost. Aircraft movement can change the relative orientation of the antennas, which is one reason why a link may perform differently during stationary testing and actual movement.

Radio-frequency interference should be distinguished from deliberate jamming, although both can produce communication loss. Jamming involves intentionally creating interference to prevent or degrade communication. A defensive assessment can study the aircraft's response to simulated loss of signal or controlled interference conditions without transmitting disruptive signals into shared spectrum. Such testing can establish whether the aircraft detects degraded communication, reports the problem, and follows its expected failsafe behavior.

Signal-quality measurements should be interpreted alongside application-level results. A radio may report acceptable signal strength while the communication protocol experiences high latency or repeated message loss. Conversely, a link may experience brief physical-layer errors that are corrected before they affect the application. Evaluating both radio measurements and message-delivery behavior provides a more complete assessment.

In a controlled laboratory, signal quality can be studied through simulated channel conditions, recorded telemetry, or appropriately shielded test equipment. The goal is to establish a relationship between defined conditions and observed performance. Measurements should be collected consistently, and the limitations of the test setup should be documented so that the results are not generalized beyond what the evidence supports.

# Authentication, Encryption, and Wireless Trust Relationships

Wireless security requires more than preventing outsiders from understanding the content of a transmission. A secure system must establish which entities are permitted to communicate, determine whether received messages are authentic, protect sensitive information where necessary, and ensure that accepted commands are appropriate for the current operating state.

Authentication establishes the identity or authority of a communicating entity according to a defined process. In a drone system, the relevant entities may include the operator's control unit, a ground station, an onboard computer, a maintenance tool, or a remote service. Authentication may use cryptographic credentials, pre-established keys, certificates, or other mechanisms appropriate to the architecture.

The existence of a pairing process does not automatically establish strong authentication. Some systems associate devices during setup but provide limited protection after pairing. Others rely on identifiers that are visible over the air or that can be copied. The security assessment should determine what evidence the receiving device requires before it accepts a connection or processes a command.

Authentication also needs to be bound to the intended role. A device authorized to send telemetry should not automatically gain permission to alter firmware or change safety-critical parameters. Similarly, a maintenance interface may require stronger authorization than an ordinary monitoring connection. Role-based permissions and clear separation of privileges reduce the impact of a compromised endpoint.

Encryption protects the confidentiality of data by making it unintelligible to parties who do not possess the required cryptographic key. It can be important for telemetry containing sensitive operational information, mission details, device identifiers, or other protected data. However, encryption alone does not necessarily establish that a message came from an authorized sender. Depending on the method used, encrypted data may still be vulnerable to modification or replay unless integrity and authenticity are also protected.

Authenticated encryption combines confidentiality with mechanisms that allow the receiver to detect unauthorized changes. When implemented correctly, it helps ensure that protected data has not been modified without detection. The receiver must still enforce appropriate rules for message freshness, permitted commands, and valid operating states. Cryptographic authenticity cannot establish that a legitimate operator's request is safe under every circumstance.

Key management is often more difficult than selecting a cryptographic algorithm. The system must generate keys appropriately, deliver them to the correct devices, store them securely, and define how they are replaced or revoked. If multiple aircraft share a single long-term secret, compromise of one device may affect the others. Where practical, individual device credentials or carefully designed group-key arrangements can limit the scope of a compromise.

Pre-shared keys can be suitable for some constrained systems, but they require a secure provisioning process and a clear policy for replacement. If the same key is embedded in every unit and can be extracted from one compromised device, the security of the entire product family may be weakened. Device-specific credentials and protected storage can reduce this risk, although they introduce additional requirements for manufacturing, maintenance, and recovery.

Certificates can support scalable identity management by associating a public key with an identity and a defined trust authority. Their usefulness depends on how devices validate certificates, manage expiration, handle revocation, and respond when trust information is unavailable. A certificate-based design that fails open when verification cannot be completed may provide less protection than expected.

Replay resistance is another essential property. A previously valid command may become inappropriate if it is accepted again after circumstances have changed. Sequence counters, timestamps, nonces, and session-specific state can help a receiver distinguish current messages from old ones. These mechanisms must be implemented consistently, including across restarts and reconnections where relevant.

The system should also consider session establishment and reconnection behavior. A temporary radio interruption may require the devices to re-establish communication. During this process, the system should not unintentionally downgrade authentication, reuse expired session state, or accept commands from an unverified source. The intended behavior should be documented and tested under controlled conditions.

Trust relationships extend beyond the radio link itself. A ground station may authenticate to a remote service, receive mission data, and forward commands to the aircraft. If the service or ground station is compromised, the radio link may still be cryptographically secure while carrying instructions that should not have been issued. Security therefore depends on the complete path from the human operator or authorized service to the flight controller.

This principle is especially important for connected aircraft that interact with cloud systems, mobile applications, or companion

computers. Each component may have its own identity, permissions, and update process. The system should avoid granting broad authority simply because one component has successfully authenticated to another. Trust should be limited to the functions necessary for each role.

Cryptographic algorithms and protocols should be selected according to current engineering guidance and the capabilities of the platform. Custom cryptographic designs are particularly difficult to evaluate because small implementation errors can undermine otherwise sound algorithms. Established protocols, supported implementations, secure key storage, and timely updates generally provide a more defensible foundation than improvised encryption schemes.

Security controls must also account for availability. Authentication failures should be reported and handled consistently, but the system must define what happens if a legitimate device loses access to required credentials or if a key becomes invalid during operation. Recovery procedures should restore authorized access without creating an unrestricted bypass around the security policy.

The most meaningful assessment examines the complete trust relationship: how identity is established, what authority follows from successful authentication, how messages are protected, how old messages are rejected, and how the system behaves when verification fails. Each of these properties contributes to wireless security, and none should be assumed merely because a device advertises encrypted communication.

# Evaluating Wireless Exposure in Shielded or Simulated Test Environments

Wireless security should be evaluated in a controlled environment that allows researchers to observe communication behavior without disrupting unrelated devices or creating unnecessary operational risk. A shielded test enclosure, a conducted radio-frequency setup, or a software-based simulation can provide an appropriate setting, depending on the question being examined. The selected method should reproduce the relevant properties of the system while remaining within the authorization and technical constraints of the research.

A shielded enclosure reduces the amount of radio energy that escapes into the surrounding environment and limits the influence of external transmissions. Its effectiveness depends on the enclosure's construction, frequency range, seams, openings, cable penetrations, and the way equipment is installed. Shielding should not be assumed to be perfect. Where radio emissions matter to the experiment, the test arrangement should be checked with suitable measurement equipment and procedures.

Conducted testing uses controlled connections between radio equipment rather than relying entirely on over-the-air propagation. Depending on the hardware, this may involve suitable attenuators, coupling components, and test instruments designed for the relevant frequencies and power levels. The arrangement can improve repeatability by reducing changes caused by antenna orientation and the surrounding environment. It also requires careful attention to the equipment's electrical limits and the manufacturer's test requirements.

Software simulation provides another useful approach. A simulated channel can introduce defined delays, packet loss, bandwidth limits, or other communication conditions without transmitting radio energy. This is particularly suitable for evaluating how the autopilot and ground station respond when messages are delayed or unavailable. However, software simulation may not reproduce all

physical-layer effects, including receiver overload, antenna mismatch, or hardware-specific radio behavior.

The test method should match the research question. If the objective is to determine whether a command is authenticated before it affects flight-control state, a software-level test may provide strong evidence. If the question concerns receiver sensitivity or antenna performance, a software simulation alone is insufficient. Selecting the correct method prevents the laboratory from claiming more than the experiment can establish.

A baseline test should establish normal communication before any conditions are changed. Researchers should record which devices are connected, which services are enabled, the expected message flow, the observed signal or channel metrics, and the normal response time. This baseline becomes the reference for evaluating later results.

Wireless exposure includes both the information that can be observed and the functions that can be reached. An outside observer might detect that a device is transmitting without being able to decode the content. Another system might decode public protocol fields but lack the credentials needed to authenticate commands. A complete assessment distinguishes passive observation, successful message interpretation, authenticated access, and the ability to influence system behavior.

These are separate security properties. The visibility of a radio transmission does not prove that the communication is insecure, just as the presence of encryption does not prove that the system is properly authenticated. Researchers should identify which property is being tested and collect evidence appropriate to it.

The laboratory can assess information exposure by examining the metadata and content available through authorized captures or simulated traffic. Relevant observations may include device

identifiers, message timing, communication frequency, unencrypted status fields, and whether sensitive information is transmitted without adequate protection. The significance of each observation depends on the system's intended use and the sensitivity of the information.

Traffic timing and volume may reveal patterns even when message content is encrypted. For example, repeated communication intervals can indicate that a device is active, while changes in traffic volume may correspond to different operating states. This does not necessarily constitute a vulnerability, but it may be relevant when assessing privacy or operational confidentiality. The assessment should establish whether such exposure matters under the system's threat model.

Authentication behavior can be evaluated using authorized test devices and known test credentials. The purpose is to verify that the receiving system accepts permitted participants and rejects participants that do not meet the defined requirements. Researchers should also assess whether authorization remains limited to the intended role and whether a failed authentication attempt produces the expected response.

Message-integrity checks can be evaluated with controlled test inputs. The test should establish whether the receiving component detects an altered message and whether the rejected input causes any unintended state change. Such experiments should use an isolated simulation or an authorized test device, rather than introducing untrusted commands into an operational aircraft.

Freshness protections can be assessed by checking how the system treats delayed, duplicated, or outdated test messages within the authorized environment. The relevant outcome is not simply whether the message is rejected, but whether the system applies its documented rules consistently. The result should be recorded

alongside the initial state, the test condition, and the observed response.

Resilience testing examines how the system behaves when the communication channel degrades. A simulated channel can introduce a known period of packet loss or delay, allowing researchers to observe whether the autopilot detects the condition, whether the ground station reports it clearly, and whether the aircraft follows its expected failsafe policy. The test should also determine whether communication recovers cleanly when normal conditions return.

Interference-related testing requires particular care. Deliberately transmitting disruptive signals into shared spectrum may affect unrelated services and may violate applicable regulations. Researchers can evaluate many availability questions using simulation, shielded arrangements, or appropriate conducted testing without creating uncontrolled interference. Any over-the-air experiment should use lawful, authorized methods and equipment appropriate to the location and frequency range.

A laboratory should also examine the limits of its own instrumentation. A receiver or capture device may miss messages because of its bandwidth, sensitivity, sampling configuration, or placement. A lack of observed traffic does not prove that no transmission occurred. Similarly, an instrument may report a signal that cannot be decoded into a valid message. These limitations should be recorded so that measurements are interpreted correctly.

Repeatability is essential when comparing wireless security controls. The same test should use the same software versions, radio configuration, initial conditions, and observation method wherever practical. If the test environment changes, the report should explain how those changes may affect the result. A result obtained in a well-shielded enclosure should not automatically be generalized to a complex outdoor environment.

The evidence record should connect each observation to a defined question. If the experiment concerns authentication, the report should establish what credentials were expected, how the test participant was identified, and what the receiver did. If it concerns link availability, the record should include the imposed condition, relevant timing measurements, observed packet delivery, and the aircraft's response. This makes the result useful to developers who need to correct weaknesses or verify existing safeguards.

The final interpretation should distinguish between the behavior of the radio hardware, the communication protocol, and the application that processes the messages. A physical-layer failure may prevent data from reaching the protocol. A protocol-level rejection may prevent a message from reaching the flight-control application. An application-level rule may reject a command even when the radio and protocol layers operate correctly. Identifying the layer responsible for the observed behavior is essential to accurate diagnosis.

A carefully controlled wireless assessment can establish whether communication boundaries behave as intended, whether unauthorized participants are rejected, whether message integrity and freshness are enforced, and whether the aircraft responds appropriately to degraded links. The strength of the findings depends on the quality of the experimental design, the reliability of the collected evidence, and the precision with which the conclusions are limited to the conditions actually tested.

# MAVLink and Drone Communication Protocols

## MAVLink Architecture, Message Structures, and System Identification

Modern unmanned aircraft depend on communication protocols to coordinate flight controllers, navigation systems, companion computers, ground control stations, and other connected components. These protocols determine how systems exchange information, how commands are represented, and how software components interpret the operational state of an aircraft. A drone may possess reliable flight hardware and carefully developed control software, yet weaknesses in its communication architecture can undermine the security of the entire platform. Understanding these relationships is therefore essential to assessing the cybersecurity of modern unmanned aircraft systems.

MAVLink is a lightweight communication protocol designed for exchanging structured messages among components of an unmanned vehicle system. It is commonly used to transmit telemetry, communicate flight status, report sensor measurements, and carry operational commands between an autopilot and a ground control station. Its design supports constrained embedded environments, where processing capacity, memory, bandwidth, and timing may be limited. Rather than requiring every component to maintain a complex, continuous connection with every other component, the protocol provides a defined message format that allows different systems to exchange information through compatible implementations.

A defining characteristic of MAVLink is its message-oriented architecture. Information is organized into discrete packets, with each message representing a particular kind of data or operation. One message may communicate the vehicle's position, another may report battery voltage, and another may convey information about a mission or flight mode. This approach allows an autopilot to transmit only the information required by a receiving component while supporting communication among systems developed independently.

The protocol's lightweight design is useful in environments where communication resources are limited. A small unmanned aircraft may exchange telemetry through a relatively low-bandwidth radio link, while a more capable platform may route messages through Ethernet, serial connections, or a companion computer. MAVLink can operate across these different transport arrangements because its message format is distinct from the physical or network medium carrying it. That separation is important when evaluating security: the protocol defines how messages are represented, but the transport determines how those messages travel between systems and which other parties might be able to access them.

MAVLink has evolved through different protocol versions, with MAVLink 1 and MAVLink 2 representing important stages in its development. MAVLink 2 introduced features that include support for message signing and extended message capabilities. However, the presence of a newer protocol version does not automatically establish that a particular installation is secure. Security depends on the version supported by each component, the configuration selected by the operator, the implementation's behavior, and the surrounding communication architecture.

A useful starting point for analysis is the distinction between a message definition and a message instance. A message definition describes the fields a particular message can contain, their data

types, their expected interpretation, and the identifier associated with the message. A message instance is the actual encoded packet transmitted at a particular moment. For example, a position-related message definition may specify fields for latitude, longitude, altitude, and associated reference information. A transmitted instance contains the values reported by the sending component at the time of transmission.

This distinction matters because a packet can be structurally valid while containing inaccurate or misleading information. A receiver may successfully decode a message without knowing whether the reported position is correct, whether the sender obtained it from a trustworthy sensor, or whether the information is sufficiently recent to support a safety-critical decision. Protocol validation must therefore address both syntactic correctness and the operational meaning of received data.

A typical MAVLink packet contains a framing indicator, a payload-length field, message identification information, identifiers associated with the sending system and component, the message payload, and a checksum. The precise layout depends on the protocol version. MAVLink 2 also supports optional signing information when signing is used. These fields allow a receiving implementation to recognize message boundaries, identify the declared message type, associate the packet with a system and component, and check aspects of packet integrity.

The system identifier and component identifier serve important organizational purposes. The system identifier distinguishes one participating system from another within a communication environment, while the component identifier distinguishes components associated with a system. An aircraft may have an autopilot, a companion computer, and additional onboard components that use distinct component identifiers. A ground control station may also participate in the message exchange under its own system identity.

These identifiers are not, by themselves, proof of identity. They are values carried in the message format, not independent credentials. A receiver that trusts a packet merely because it declares an expected system identifier has confused identification with authentication. This distinction is one of the most important principles in MAVLink security. Identifiers help organize communication, but trust must come from additional controls appropriate to the system's threat model.

Message identifiers provide another layer of organization. Each message type has an associated identifier that tells the receiver how to interpret the payload. The receiver uses the relevant message definition to decode the fields and make them available to application software. If a system does not support a particular message definition, it may be unable to interpret that message correctly. Compatibility problems can arise when components use different dialects, different generated libraries, or different revisions of message definitions.

A MAVLink dialect is a collection of message definitions and related protocol information used by a particular implementation or application environment. Different systems may share a common foundation while supporting additional messages for specific hardware or operational requirements. This flexibility allows the protocol to accommodate a wide range of platforms, but it also introduces a configuration-management concern. A system must know which definitions its components support and which messages they are permitted to use.

Poorly managed dialect differences can produce inconsistent interpretations of messages. One component may recognize a field or command that another component does not support. An application may decode a message correctly but assign it an operational meaning that differs from the sender's intention. Such inconsistencies can affect telemetry displays, mission execution, and security

monitoring. Maintaining compatible message definitions and testing them across software versions reduces the risk of these failures.

Message structure also influences how a system handles incomplete, malformed, or unexpected input. A robust receiver should verify framing, length, message identification, and checksum information before passing decoded content to higher-level application logic. It should also account for unsupported message identifiers and unexpected field values. Successful decoding must not be treated as authorization to execute an operation.

The checksum used by MAVLink helps detect certain transmission errors and supports verification that a packet has not been corrupted in transit. It is not a cryptographic authentication mechanism. A party capable of constructing a new packet can also calculate the appropriate checksum. Consequently, checksum validation alone cannot establish that a message originated from a trusted sender or that it has not been deliberately altered and reconstructed.

System identification must therefore be evaluated separately from packet validation. A security assessment should establish which systems and components are expected to participate, what identities they use, which communication paths connect them, and what operations each participant is permitted to perform. This includes documenting the autopilot, ground control station, companion computer, telemetry radios, network bridges, and any external services that exchange flight information.

The resulting system inventory provides the foundation for a meaningful trust model. A component that reports battery status does not necessarily require permission to change flight modes. A monitoring computer may need access to telemetry without receiving authority to issue navigation commands. A maintenance workstation may need diagnostic access while the aircraft is disarmed but should not retain unrestricted control during normal operation. These

distinctions should be expressed through system design and enforced by the components responsible for accepting commands.

An effective assessment also considers the difference between the identity asserted by a message and the identity established by the communication environment. A message may declare a known system identifier even when it arrives through an unexpected network interface. The receiving application should not assume that every packet bearing a familiar identifier is trustworthy. Network segmentation, authenticated connections where supported, controlled routing, message signing, and command authorization can provide additional assurance.

Timing is another important aspect of system identification. Telemetry may be accurate when generated but unsuitable for a decision if it arrives too late. A receiver should consider message frequency, timestamps where available, sequence information, and expected update intervals. Some message types have no independent timestamp in their payload, so the receiver may need to record arrival times and assess freshness using the application's operational requirements.

For example, a ground control station displaying the most recent position report may continue to show an old position if communication has been interrupted. Without a clear indication that the telemetry is stale, an operator may interpret the display as current. The same issue can affect automated monitoring systems that rely on telemetry to determine whether an aircraft is following its expected route. A secure implementation must therefore consider availability, freshness, and confidence in addition to packet structure.

# Telemetry Routing Between Autopilots and Ground Control Stations

Telemetry routing determines how messages move between the aircraft and the systems used to monitor or control it. In a simple installation, an autopilot may communicate with a ground control station through a direct serial connection and a pair of telemetry radios. In a more complex arrangement, a companion computer, network bridge, or communication service may forward messages between several endpoints. Each additional component creates another dependency that must be understood and secured.

The autopilot is often the primary source of flight-state information. It generates or forwards messages describing position, altitude, attitude, speed, battery condition, sensor health, flight mode, and mission progress. The ground control station receives these messages and presents them to an operator through displays, alerts, maps, and mission-planning interfaces. Depending on the platform and configuration, the ground control station may also send commands that alter the aircraft's behavior.

The communication path between these components is not necessarily a single uninterrupted connection. A message may travel from the flight controller through a serial interface, pass through a telemetry radio, enter a computer connected to a network, and then be forwarded to an application. In another design, a companion computer may collect data from the autopilot and distribute it to several applications. The security characteristics of the complete route depend on each stage rather than solely on the protocol used at the beginning or end.

Routing components require particular attention because they can influence which messages reach which recipients. A router may

forward telemetry to a monitoring application, relay commands from a ground control station, or distribute selected messages to multiple endpoints. If forwarding rules are too permissive, a component intended only to observe telemetry may gain access to messages intended for a control interface. A poorly isolated endpoint may also receive commands it was never intended to process.

The distinction between a router and an application is important. A router generally forwards messages according to its configuration and routing logic. An application interprets messages and may act on their contents. A component that forwards traffic without understanding its meaning cannot necessarily determine whether a command is appropriate for the aircraft's current state. Security controls should therefore exist at the points where decisions are made, not only at intermediate forwarding components.

Consider a system in which a telemetry router connects an autopilot to a ground control station and a separate engineering workstation. If the workstation requires only read-only access to flight information, the routing design should prevent it from sending operational commands to the autopilot. Simply placing the workstation on the same network and instructing its user not to issue commands is not a reliable security boundary. The system should enforce the intended restriction through permissions, interface separation, filtering, or other controls supported by the implementation.

Routing rules must also account for broadcast behavior. Some message-based systems distribute telemetry to multiple recipients rather than maintaining a unique conversation between two endpoints. This can be useful for monitoring and coordination, but it increases the importance of controlling who can access the communication medium. If every connected endpoint can observe all traffic, sensitive operational information may be exposed to systems that have no legitimate need to receive it.

Operational information may reveal more than the current position of an aircraft. Mission-related messages can disclose intended routes, operating schedules, geographic boundaries, or patterns of activity. Status reports may reveal when an aircraft is armed, when it is experiencing a fault, or when an operator is attempting to establish a connection. The sensitivity of these details depends on the mission, but a security review should identify which data requires protection and whether the transport or network provides that protection.

Telemetry routing also affects availability. A router that becomes overloaded, stops forwarding packets, or enters an unexpected state can interrupt communication between the autopilot and its operator. Excessive traffic can delay time-sensitive updates or consume processing resources required for normal forwarding. Although a well-designed autopilot should retain appropriate onboard control when external telemetry is lost, the loss of communication can still reduce situational awareness and limit an operator's ability to intervene.

A routing assessment should document the expected message flow for each operational mode. Normal flight, preflight configuration, maintenance, mission upload, and post-flight review may require different permissions and communication paths. A connection that is acceptable during bench testing may be inappropriate during flight. Likewise, a maintenance port that remains available after servicing may create an unnecessary route into a system that is otherwise isolated.

Network segmentation can reduce exposure by separating flight-control traffic from administrative services, general-purpose computing, and unrelated devices. Segmentation does not itself authenticate MAVLink messages, but it can limit which systems can reach the interfaces that carry them. Firewalls, interface binding, access-control rules, and explicit routing policies can further reduce unnecessary connectivity. The exact measures depend on whether

the platform uses serial links, IP networking, wireless bridges, or a combination of transports.

A ground control station should also distinguish between telemetry reception and command transmission. An operator may need to monitor several aircraft simultaneously while issuing commands to only one. The interface should make the selected target unambiguous and provide clear feedback about the system receiving each command. A mistake in system selection can be operationally significant even when all messages are technically valid.

Where several aircraft share a communication environment, routing policy should be designed to prevent unintended cross-system command delivery. System identifiers help receivers distinguish participants, but they should not be the only safeguard. The network design, application behavior, and command authorization mechanisms must work together to ensure that a command intended for one aircraft cannot be accepted by another merely because both are reachable through the same router.

Routing behavior should be tested under realistic failure conditions. These include temporary link loss, delayed packets, duplicate delivery, router restart, partial network failure, and recovery after an interrupted connection. A secure system should recover predictably, avoid interpreting stale telemetry as current, and avoid replaying queued commands without checking whether they remain appropriate. Recovery logic is especially important when commands depend on a specific vehicle state.

Monitoring should record significant routing and connection events without creating an excessive processing burden. Useful records may include endpoint connections, changes to routing configuration, rejected messages, authentication failures, and interruptions in telemetry flow. Logs should avoid exposing signing keys or other secrets. They should also use reliable timestamps where possible so

investigators can reconstruct the order of events after a fault or suspected security incident.

## Message Signing, Authentication, and Protocol Configuration

Message signing provides a way to verify that a signed MAVLink packet was generated by a participant possessing the relevant secret key and that the signed content has not been altered without detection. MAVLink 2 supports message signing, but signing must be configured correctly and consistently across the participating components. A system that supports signing in principle may still accept unsigned messages if its configuration permits them.

Signing is particularly important because the system and component identifiers carried in a packet are not secret. An observer can learn these identifiers from ordinary communication. Without a suitable authentication mechanism, a receiver that relies only on declared identifiers and checksums may have no reliable way to distinguish an authorized sender from an untrusted party constructing similar messages.

MAVLink 2 signing uses a secret key shared by participating components and a signing structure that includes information used to authenticate the packet and help identify repeated or out-of-order transmissions. The authentication value is derived from the message and signing information using the protocol's specified cryptographic procedure. Receivers that enforce signing can validate the authentication value before accepting a packet as signed traffic.

The security value of this mechanism depends on how the key is generated, distributed, stored, and protected. If the same key is

installed in many unrelated systems, a compromise of one device may affect the trustworthiness of the broader group. If the key is stored in an exposed configuration file or can be read through an unrestricted debugging interface, signing may provide less protection than expected. A sound key-management process limits access to secrets and defines how keys are replaced when a component is lost, compromised, or retired.

Message signing should not be confused with encryption. Signing helps establish message authenticity and integrity, but it does not inherently conceal the packet's contents. Depending on the transport and network design, an observer may still be able to read telemetry, identify message types, and infer operational details. If confidentiality is required, it must be provided through an appropriate protected transport or another suitable security layer. The design must account for the capabilities and limitations of the particular links in use.

Signing also does not establish that the sender is authorized to perform every operation represented by a message. A valid signature indicates that the packet passed the configured authentication check under the relevant key and protocol rules. It does not prove that the command is appropriate, safe, or permitted in the current flight state. If a trusted component is compromised, it may generate correctly signed messages that are nevertheless harmful. Authentication and authorization therefore solve different problems and must be treated separately.

Replay protection is another important consideration. A recorded packet may be authentic and unmodified yet no longer be appropriate when received again. MAVLink 2 signing incorporates link and timestamp-related information that supports rejection of certain replayed or out-of-order signed messages. However, replay resistance depends on correct implementation and state management. Receivers must follow the protocol's validation rules,

maintain relevant signing state, and handle restarts or clock-related conditions according to the implementation's documented behavior.

A system should be tested to determine how it behaves when signing is enabled, when an unsigned message arrives, when a signature is invalid, and when a packet appears outside the acceptable signing sequence. The expected behavior should be explicit. In a configuration that requires signed commands, an unsigned command should not be accepted merely because its message identifier and checksum are valid. At the same time, a system must avoid an inconsistent configuration in which one component expects signed traffic while another silently relies on unsigned messages.

Mixed configurations are a common source of uncertainty. A flight controller may support signing while a ground control station or telemetry router is configured differently. Some components may forward signed messages without validating them; others may verify signatures before passing traffic onward. These differences can complicate troubleshooting and may create gaps in the intended trust boundary. Every participating component should have a documented role in the signing architecture.

Configuration management is therefore part of protocol security. Operators and developers should maintain a record of protocol versions, message dialects, signing policies, key identifiers or key-management references, and relevant software revisions. Secrets themselves should not be copied into ordinary documentation or logs. Configuration records should instead describe where keys are managed, which components are authorized to use them, and how rotation or revocation is performed.

The configuration should also define whether signing is mandatory for all traffic or only for selected operational functions, where the implementation supports such distinctions. A policy that protects only some commands may be appropriate in a carefully designed system, but it should not be mistaken for universal message

authentication. Any accepted unsigned traffic must be understood, justified, and restricted to the minimum necessary scope.

Testing should include configuration drift. A replacement ground control station, updated companion computer, or newly installed telemetry radio may introduce different defaults or omit a required security setting. The aircraft may appear to function normally while its authentication assumptions have changed. A controlled deployment process should verify the intended configuration after updates and hardware replacement, not merely assume that the previous security posture remains intact.

Key storage deserves attention throughout the component lifecycle. Embedded devices may expose debugging ports, removable storage, maintenance interfaces, or configuration backups that contain sensitive information. Access to those facilities should be restricted according to the operational threat model. When devices are transferred, repaired, or decommissioned, the handling process should prevent secrets from remaining accessible to unauthorized parties.

Security configuration should be understandable to the people responsible for operating the aircraft. Unclear warnings, ambiguous interface labels, and silent fallback behavior can cause an operator to believe that communication is authenticated when it is not. The system should clearly report whether the expected signing policy is active and whether authentication failures are occurring. It should not expose secret material as part of this reporting.

# Command Validation, Authorization, and Secure Message Processing

A command received through a drone communication protocol can influence flight mode, mission execution, navigation behavior, payload operation, or other aspects of system activity. The consequences depend on the command, the aircraft's capabilities, and its current state. Secure processing must therefore begin with the principle that receiving a well-formed message is not equivalent to receiving permission to execute it.

Command handling should be organized as a sequence of distinct checks. The implementation first determines whether the packet can be parsed and whether its framing, length, and integrity information are valid. If signing is required, the receiver then verifies the message authentication information according to the protocol rules. The application evaluates the sender's permissions, checks whether the requested operation is supported, and determines whether the command is valid under the aircraft's current operational conditions.

The order and precise placement of these checks depend on the software architecture, but the security objectives remain consistent. Invalid packets should not reach sensitive command-processing logic. Unauthenticated messages should not be allowed to bypass the configured trust policy. Authenticated senders should not automatically gain authority over every operation. Commands that conflict with safety constraints should be rejected or handled through a defined safe response.

Authorization is especially important when a platform has several communicating components. A ground control station may be authorized to upload a mission, while a monitoring application should be limited to receiving telemetry. A payload computer may need to report sensor status but should not be permitted to change the aircraft's navigation configuration. These roles should be enforced by software and interface controls rather than left to informal expectations about how each component will behave.

The same principle applies to human operators. A maintenance interface may legitimately expose configuration functions that should not be available during ordinary operations. A system can reduce risk by requiring explicit operating modes, disarmed-state checks, or additional confirmation before allowing sensitive changes. Such safeguards must be carefully designed so that they do not create new hazards or interfere with necessary emergency procedures.

Command validation should include range checks, type checks, state checks, and consistency checks. A numeric parameter may be represented in the correct data type yet fall outside the range supported by the aircraft. A navigation command may contain valid coordinates but request a route that violates configured operational boundaries. A command to change flight mode may be syntactically correct but incompatible with the aircraft's current sensor health or control state.

Validation should use the constraints appropriate to the specific vehicle rather than relying only on generic protocol definitions. A message specification may define the structure of a parameter without establishing every operational limit for every aircraft. The application must apply platform-specific rules and communicate rejection reasons clearly enough to support troubleshooting without revealing sensitive implementation details.

Secure message processing also requires careful handling of unsupported or unexpected messages. A receiver should not assume that every message identifier it encounters is recognized or safe to process. Unknown message types should be handled predictably, and unsupported commands should not trigger unintended fallback behavior. Software should avoid treating malformed parameters as default values if doing so could cause an unexpected operation.

Robustness against resource exhaustion is another concern. Excessive traffic, repeated invalid packets, or unusually large

volumes of telemetry can consume CPU time, memory, and communication bandwidth. A parser should enforce the length constraints defined by the protocol and avoid unsafe memory operations. Applications should use sensible limits on queues and processing workloads, particularly when running on constrained embedded hardware.

Queue management matters because commands and telemetry may have different timing requirements. A delayed status update may be less harmful than a delayed flight-control command, while an old mission-related command may become unsafe if executed after the aircraft's state has changed. Systems should not assume that every queued message remains valid indefinitely. Where commands can be delayed, duplicated, or retried, the application should define how it identifies completion, prevents unintended repeated execution, and handles commands that expire or become inappropriate.

The distinction between command acknowledgement and successful completion must also be clear. A protocol response may indicate that a command was received or accepted for processing, but that does not necessarily mean the requested operation has completed. For example, a system may acknowledge a request to change a mode before the aircraft has met all conditions needed to enter that mode. The ground control station should interpret acknowledgements according to their defined meaning and monitor subsequent status messages when confirmation of the resulting state is required.

Error handling must be designed as part of the security architecture. If a command fails validation, the system should reject it without corrupting internal state. If a dependency is unavailable, the system should avoid assuming that an operation succeeded. If a communications link is lost during a command sequence, the application should follow a documented recovery policy instead of replaying every pending request automatically.

Logging can help developers and operators understand rejected commands, malformed traffic, authorization failures, and unexpected state transitions. Logs should include enough context to reconstruct relevant events while minimizing the collection of sensitive information. They should not store secret signing keys, credentials, or unnecessary personal data. Where the environment supports it, access to logs should be restricted and changes to important records should be detectable.

Secure processing must also account for the possibility that a trusted application contains a software defect. A correctly authenticated command can still exploit a flaw in parameter handling, state management, or an underlying library. Secure development practices therefore include code review, static analysis where appropriate, dependency management, testing of parser boundaries, and regression tests for previously discovered defects. Protocol security cannot compensate for unsafe implementation logic.

Safety and cybersecurity are closely related but are not identical. A command may be authorized from a cybersecurity perspective and still create an unsafe condition if the aircraft's sensors are unreliable or its environment has changed. Conversely, a safety mechanism may prevent an operation even when the sender is fully authenticated. A robust design combines access control with independent operational constraints and ensures that security checks do not disable essential safety functions.

## Testing Protocol Reliability With Simulators and Authorized Test Endpoints

Testing MAVLink communication in a controlled environment allows developers and security professionals to examine message handling, routing, authentication, and recovery behavior without exposing an operational aircraft to unnecessary risk. A simulator can provide a virtual vehicle and a representative autopilot environment, while a test endpoint can act as a ground control station, telemetry consumer, or command source. Together, these tools support repeatable experiments and make it easier to distinguish a software defect from a hardware or radio problem.

A well-designed test environment begins with a clear scope. The assessment should identify which software components are being evaluated, which message types are relevant, which communication paths are included, and which behaviors are expected. The test should use systems the researcher owns or has explicit permission to assess. Production aircraft, third-party networks, and live radio environments should not be included merely because they are accessible.

Simulation is particularly useful for examining state-dependent behavior. A test can evaluate how an autopilot reports its flight mode, responds to a valid command, handles a command that is inappropriate for the current state, or behaves when telemetry becomes unavailable. Because the environment is controlled, the same conditions can be reproduced after a code change. This supports regression testing and provides evidence that a security correction addresses the original issue without introducing a new fault.

A simulator does not reproduce every property of a physical aircraft. Real sensors introduce noise, hardware interfaces may behave differently under load, and radio links can experience interference or fading that a simplified simulation does not fully represent. Timing characteristics, processor constraints, and peripheral behavior may also differ. Simulator results should therefore be treated as evidence

about the components and conditions actually represented, not as proof that every real-world configuration is secure.

Test endpoints provide another means of evaluating protocol behavior. An authorized endpoint can generate or receive expected telemetry, record packet timing, verify message decoding, and observe how the system handles authentication or configuration errors. A passive monitoring endpoint is particularly useful for determining whether messages are routed to unintended recipients. Its permissions should be limited so that the act of monitoring does not inadvertently create a new command path.

The first stage of protocol testing should establish a baseline. The test team records the software versions, protocol version, relevant message definitions, transport configuration, signing policy, and expected communication rates. Normal operation is then observed under representative conditions. This baseline helps identify whether later anomalies result from the test itself, an environmental change, or a genuine defect in the implementation.

Structural validation can be assessed using malformed or incomplete test inputs generated within the controlled environment. The objective is to determine whether the receiver rejects invalid framing, inconsistent lengths, unsupported message identifiers, and incorrect integrity values without crashing or entering an unexpected state. These tests should focus on the receiver's robustness and should not involve transmitting unapproved traffic to unrelated systems.

The testing process should verify that a rejected packet does not produce an unintended side effect. For instance, an invalid command should not alter the selected flight mode, modify a mission, or change configuration values. The test should also examine whether the receiver continues processing legitimate traffic after a rejection. Robust software should recover cleanly from malformed input rather than becoming unresponsive after a single error.

Authentication testing should verify the behavior prescribed by the configured security policy. In a test system that requires signed messages, the assessment can confirm that correctly signed traffic is processed as expected and that unsigned or incorrectly signed traffic is rejected where required. The test should also examine the system's response to replay-related conditions using the implementation's supported test facilities. Results should be compared with the protocol specification and the actual configuration rather than judged against assumptions about default behavior.

Configuration testing should include deliberate differences between components. For example, a test may verify how the system reports a mismatch between the signing policy of the ground control station and the autopilot. It may also confirm that a replacement component does not silently reintroduce a permissive configuration. Such tests are important because a system can pass an initial security assessment and later become weaker through routine maintenance or software updates.

Routing tests should confirm that messages reach only the intended endpoints. A test environment can include a telemetry receiver, a command-capable endpoint, and a monitoring component with restricted permissions. The assessor can then verify that the monitoring component receives the information required for its role without gaining the ability to issue commands. The assessment should also consider whether routing rules remain effective after a component reconnects or restarts.

Reliability testing should examine delayed, duplicated, missing, and reordered messages where these conditions can be reproduced safely. The aim is to understand how the application handles imperfect delivery. Telemetry consumers should identify stale information rather than presenting it indefinitely as current. Command-processing logic should not repeat non-idempotent operations simply because an acknowledgement was delayed.

Recovery behavior should be consistent with the aircraft's documented operational and safety requirements.

Load testing can assess whether a system remains responsive when receiving traffic near its expected operating limits. The test should remain within an approved environment and use controlled workloads that do not interfere with unrelated systems. Useful observations include CPU utilization, memory consumption, queue growth, packet loss, response latency, and the ability to continue processing legitimate messages. Results should be interpreted in relation to the platform's real-time requirements and available resources.

A communication interruption is another valuable test condition. The test team can simulate the loss of a telemetry connection, observe the aircraft's configured behavior, and determine how the ground control station indicates the interruption. After the link returns, the assessment should verify whether the system restores communication predictably, whether old information is marked as stale, and whether pending commands are handled according to policy. Any automated recovery process should be evaluated for unintended state changes.

Testing should also examine software updates and dependency changes. A revised protocol library may alter parsing behavior, message support, or signing compatibility. A new ground control station version may change the way it handles acknowledgements or connection failures. Regression tests should cover the message types and security assumptions that are essential to the platform, with particular attention to components that bridge different transports or run with elevated privileges.

The quality of the assessment depends on its records. Each test should document the objective, environment, configuration, input conditions, expected behavior, observed result, and any relevant logs. Findings should distinguish confirmed defects from suspected

weaknesses and should state the conditions under which the behavior was observed. This avoids presenting a narrow laboratory result as evidence of a universal vulnerability.

A useful finding explains both the technical issue and its operational significance. A parser that accepts a malformed packet may represent a robustness problem, but its severity depends on whether the packet can reach the affected component, what the parser does with the data, and whether the condition can influence flight behavior. Similarly, an unsigned message being accepted is a meaningful concern only when the system's intended policy requires that message to be authenticated or when the acceptance creates an unauthorized capability.

Remediation should be followed by repeat testing. A configuration change may resolve an authentication issue while unintentionally blocking required telemetry. A parser correction may eliminate a crash but introduce excessive processing overhead. Regression testing verifies that the security improvement works as intended and that essential communication functions remain reliable. Where possible, the same controlled test conditions should be used before and after the change so that results can be compared directly.

The final assessment record should identify the tested components and versions, the limits of the test environment, the findings, and the corrective actions taken. It should state which behaviors were verified and which remain untested. This level of precision helps engineers, operators, and maintainers make informed decisions without overstating the assurance provided by a simulator or a limited set of test endpoints.

# Ground Control Stations and Operator Security

## Ground Control Software Architecture and Aircraft Communication

A ground control station serves as the primary interface through which an operator observes, configures, and manages an unmanned aircraft system. Its responsibilities extend beyond displaying the aircraft's position or providing controls for takeoff and landing. Depending on the platform, ground control software may support mission design, flight configuration, telemetry interpretation, equipment calibration, log analysis, software updates, and communication with supporting systems. Because these functions connect human decisions with automated flight operations, the security of the ground control station is an essential part of the security of the entire aircraft system.

The ground control station is best understood as a collection of cooperating software components rather than a single application. A typical installation may include a graphical interface, a communication module, a telemetry parser, a mission-planning engine, a parameter-management service, mapping functionality, local storage, and interfaces to external devices. Some applications combine most of these functions within one process, while others separate them into services or supporting applications. The particular arrangement depends on the aircraft platform, the operating system, and the operational requirements.

This architecture creates several distinct trust boundaries. The graphical interface receives information from the operator and presents data received from the aircraft. The communication layer handles messages arriving through serial ports, telemetry radios, or network connections. The parsing and application layers interpret those messages and may use them to update displays, store records, or initiate actions. A configuration component may translate an operator's selection into commands sent to the autopilot. A logging component may retain information for later review. Each boundary represents a point where incorrect assumptions, unsafe input handling, or excessive permissions can affect the wider system.

A ground control application must process information originating outside its own process. Telemetry may come from an onboard flight controller, a companion computer, a network bridge, or a remote endpoint. Even when the aircraft is operated by a trusted organization, the communication path may cross wireless links, shared networks, or intermediate equipment that is not under the direct control of the application developer. The software should therefore treat incoming information as external input and validate it before relying on it.

The same principle applies to messages that appear to originate from a familiar aircraft. An identifier displayed by the application does not necessarily establish that the sender is authentic. The application should distinguish between identifying a participant, verifying the origin and integrity of a message, and determining whether the participant is authorized to perform a particular operation. These are separate security questions, and an effective architecture addresses each one explicitly.

Communication modules require special care because they often connect the ground control station to operationally important functions. A module that receives telemetry may also be capable of transmitting commands, depending on its design and permissions. If

the application uses a single communication channel for both monitoring and control, the security boundary between those activities may be weaker than expected. Where practical, the architecture should distinguish read-only monitoring from command-capable functions and restrict command transmission to the components that require it.

The communication layer should also handle interrupted connections predictably. Wireless links can experience interference, temporary loss of coverage, equipment restarts, or changes in network availability. When telemetry stops arriving, the application must not continue presenting old values as though they represent the aircraft's current state. Position, battery status, flight mode, and other time-sensitive information should be associated with suitable freshness indicators. An operator needs to understand whether a value is current, delayed, or no longer being updated.

A stale-data problem is not merely a display defect. An operator who believes the aircraft is still following an expected route may delay an intervention if the displayed position has not changed because communication has failed. Similarly, an outdated battery reading can create a false impression of available flight time. The ground control station should communicate loss of information clearly and avoid implying certainty when the underlying data is no longer reliable.

The application must also distinguish between command transmission, command acknowledgement, and successful completion. Sending a command does not establish that the aircraft received it. Receiving an acknowledgement does not necessarily prove that the requested action has completed. A flight controller may accept a request but remain unable to perform the operation because a safety condition has not been satisfied. The ground control station should interpret responses according to their defined meanings and, when necessary, verify the resulting aircraft state through subsequent telemetry.

This distinction is particularly important for mission changes, flight-mode transitions, calibration, and other operations that affect aircraft behavior. An interface should show whether an action is pending, accepted, rejected, or confirmed complete whenever the underlying protocol supports those distinctions. Ambiguous feedback can lead operators to repeat commands, select conflicting settings, or assume that an operation succeeded when it did not.

Ground control software also depends on accurate handling of aircraft identity. An operator may work with several vehicles during a single session, and some systems permit multiple aircraft to appear within the same communication environment. The interface should make the active vehicle unmistakable and show which aircraft will receive a proposed command. Where the consequences of a mistaken selection are substantial, confirmation should identify the target clearly rather than relying on a small icon or a previously selected window.

The software architecture should account for the privileges required by each component. A mapping interface does not normally need unrestricted access to operating-system administration functions. A telemetry display should not require permission to modify unrelated system files. A log viewer should not need to alter flight configuration simply to read stored records. Restricting permissions reduces the consequences of a software defect or compromise in a component that does not require broad authority.

Process separation can help enforce these boundaries, although it is not a substitute for sound design. A separate process may still hold excessive permissions, communicate over an unrestricted local interface, or share sensitive files with other applications. The relevant question is whether each component has only the access required for its role and whether communication between components is validated and controlled.

A secure ground control station must also remain dependable under ordinary operating pressure. Operators may need to react quickly to warnings, communication interruptions, or unexpected aircraft behavior. Interfaces should avoid concealing critical alerts, presenting contradictory status indicators, or requiring unnecessary navigation through menus during time-sensitive situations. Security controls should support sound operational decisions rather than create confusion that encourages users to bypass them.

# Mission Planning, Flight Configuration, and Telemetry Management

Mission planning connects an operator's intentions with the instructions that an aircraft will execute. A mission may contain navigation points, altitude constraints, speed settings, actions associated with particular locations, and other platform-specific parameters. The ground control station translates this information into a representation that the aircraft's autopilot can interpret. Because the resulting mission may guide an aircraft with limited direct human intervention, the integrity of mission data is a major security and safety concern.

Mission files should be treated as operational records, not as harmless documents. They can contain coordinates, route sequences, altitude values, timing information, and actions intended for onboard systems. Depending on the mission and platform, they may reveal sensitive facilities, inspection routes, survey areas, operational schedules, or recurring patterns of activity. Unauthorized changes can also affect where the aircraft travels and what actions it performs.

The planning process should preserve a clear relationship between the mission approved by the operator and the mission actually loaded onto the aircraft. A mission may be designed on one workstation, transferred to another, reviewed by a supervisor, and uploaded later through a different connection. Each transfer creates an opportunity for version confusion, accidental modification, or use of an outdated file. A controlled process should identify the approved version and make subsequent changes traceable.

Mission review should include both technical validity and operational suitability. A file can be syntactically correct yet contain coordinates outside the intended operating area. A route may use valid altitude values but conflict with local restrictions or the aircraft's capabilities. A mission may include an action that is supported by the software but inappropriate for the planned activity. The ground control station should validate available constraints and make relevant warnings visible before upload.

Automated validation can detect many common errors, but it cannot replace competent human review. The software may know that a waypoint is valid while lacking sufficient context to determine whether the location is appropriate. It may verify that a parameter falls within a supported range without knowing whether that value is suitable for the planned conditions. Mission validation should therefore combine software checks with review procedures proportionate to the risks of the operation.

Flight configuration requires similar discipline. Parameters may determine how the aircraft responds to operator input, interprets sensors, manages power, handles navigation, or reacts to failures. An incorrect setting can have consequences that are not immediately visible on the ground. For example, a configuration change may appear to save successfully while affecting the behavior of a later flight mode or a safety-related response.

Configuration management should distinguish between the settings required for normal operation, temporary adjustments made for a particular mission, and changes made for maintenance or testing. Without this distinction, temporary values can become permanent defaults, or settings intended for one aircraft may be copied to another with different hardware. A configuration record should identify the aircraft, software version, relevant hardware, and purpose of the change.

Before applying configuration changes, the ground control station should provide enough context for the operator to understand their significance. Parameters should have meaningful labels and units, and values should be validated against known constraints where possible. Changes affecting safety-critical behavior should receive appropriate review rather than being treated as routine interface preferences. The application should also make it clear whether a value has been saved locally, transmitted to the aircraft, or confirmed by the receiving system.

Telemetry management is closely connected to these responsibilities. Telemetry allows the operator to observe the aircraft's state and assess whether it is behaving as expected. However, telemetry is not a perfect representation of reality. Measurements may be delayed, incomplete, noisy, incorrectly interpreted, or affected by a sensor fault. The ground control station should present the available information with appropriate context rather than implying that every reported value is equally reliable.

Different telemetry fields have different operational meanings and update requirements. Position may be essential for navigation awareness, while battery voltage and current can help an operator evaluate available power. Attitude information may be useful for observing flight behavior, and status messages may indicate sensor faults, navigation limitations, or changes in flight mode. The application should preserve the distinctions between these data types

and avoid combining them into an oversimplified status indicator that hides important exceptions.

Telemetry quality should be evaluated through freshness, consistency, and plausibility. A position report that has not changed for an unusually long period may indicate a stationary aircraft, but it may also reflect a communication interruption. A battery measurement that changes abruptly may represent a genuine electrical event or a faulty reading. A flight mode displayed in the interface should be consistent with the latest accepted status information rather than merely reflecting the operator's most recent request.

When information is missing or contradictory, the application should communicate uncertainty. A warning that telemetry is unavailable is more useful than a display that quietly retains old values. Where different sources provide overlapping information, the application should define which source takes precedence and how disagreements are reported. These decisions should be documented and tested rather than left to accidental behavior within the interface.

Mission uploads and configuration changes should also be protected against interruption. If a transfer fails halfway through, the system should avoid treating the incomplete operation as successful. Where the platform supports verification, the ground control station should confirm that the expected mission or configuration reached the aircraft and that the resulting state matches the approved version. If verification is unavailable, the limitation should be made clear to the operator.

The same care applies when reconnecting after a communication failure. The ground control station should not assume that its local mission or parameter values still match the aircraft. The vehicle may have continued operating, received a change through another authorized station, or restarted with different settings. A

reconnection procedure should establish the current state before presenting local information as authoritative or transmitting pending changes.

Mission and telemetry records should support later reconstruction of events. Where operationally appropriate, the system should record when a mission was created, reviewed, uploaded, and changed, together with the relevant aircraft identity and software version. Telemetry logs should retain enough information to support troubleshooting and incident analysis. Records should be protected against unauthorized modification and retained according to the organization's operational, legal, and privacy requirements.

## Workstation Services, Network Interfaces, and Software Dependencies

A ground control station is only as secure as the workstation and supporting services on which it depends. Even well-designed flight software may be exposed by an insecure operating system, an outdated library, an unrestricted network interface, or a compromised supporting application. The workstation should therefore be treated as an operational system with a defined security purpose rather than as an ordinary general-use computer that happens to run flight software.

The operating system provides process isolation, file permissions, network access, device management, and other services used by the ground control application. If the operating system is compromised, an attacker may be able to observe telemetry, modify mission files, access stored credentials, or interfere with the application. Keeping the operating system supported and updated is consequently an important part of ground control security.

Updates should be planned and verified rather than applied without regard to operational requirements. A security update may correct a serious vulnerability, while a major application update may change message compatibility, configuration behavior, or support for a particular aircraft. Organizations should maintain an update process that evaluates security advisories, tests relevant functionality, and records the versions approved for operational use. Delaying updates indefinitely increases exposure, but deploying untested changes immediately before an important mission can also create avoidable risk.

Workstations should run only the software required for their assigned role. General browsing, personal communications, entertainment applications, and unrelated utilities can increase the number of components that may introduce vulnerabilities or handle untrusted content. A dedicated ground control workstation reduces some of this exposure and makes the system easier to inventory, maintain, and monitor.

The degree of separation should reflect the mission's sensitivity. A training workstation may reasonably use a different security profile from a computer that controls aircraft during critical infrastructure inspections. In either case, the organization should understand which applications are installed, which services are active, and which users can change system settings. Unnecessary services should be disabled, and administrative privileges should be limited to people who require them.

Network interfaces require particular attention. A workstation may connect simultaneously to telemetry equipment, an organizational network, wireless access points, removable devices, and external services. Each connection creates a potential route through which information can enter or leave the system. A network interface used to reach the aircraft should not automatically provide unrestricted access to unrelated systems, and general-purpose network traffic

should not be allowed to interfere with flight-related communications.

Where possible, the ground control workstation should use network rules that restrict communication to the endpoints and services required for operation. Unnecessary inbound connections should be blocked, and outbound access should be limited where practical. Local applications that expose management interfaces should require appropriate authentication and should not listen on every network interface unless that behavior is necessary and deliberately configured.

The distinction between a network connection and an authorized communication path is essential. A device may be physically connected to the same network as the ground control station without having a legitimate reason to reach its command interface. Network segmentation, host firewalls, restricted service bindings, and controlled routing can help limit exposure. These controls should be tested after configuration changes to verify that intended access remains available while unnecessary paths are blocked.

Wireless networking adds further considerations. An operator may use a wireless connection for telemetry while the workstation also connects to an office network or a mobile hotspot. Weak network security, shared credentials, or automatic connections to untrusted networks can expose the workstation to risks unrelated to the aircraft itself. Operational procedures should specify which networks may be used and how the workstation should behave when the preferred connection is unavailable.

Software dependencies are another important source of risk. Ground control applications often rely on libraries for message parsing, mapping, graphics, cryptography, serial communication, networking, and data storage. A vulnerability in a dependency can affect the application even when its own code is well written. Organizations

should maintain an inventory of important dependencies and use supported versions with a known maintenance path.

Dependency management should include more than checking whether a package has a newer version. The organization needs to understand which components are included in the deployed application, how they are obtained, and whether updates can be verified. Software packages should come from trusted distribution channels, and integrity checks should be used where available. Unofficial builds or modified installers can undermine confidence in the software before it is even launched.

The software supply chain also includes plugins, extensions, scripts, configuration bundles, and auxiliary tools. A mapping extension or diagnostic utility may have access to local files or network connections that exceed its apparent purpose. Such components should be reviewed before deployment, especially when they originate outside the organization's normal software-management process. Unnecessary extensions should be removed, and the use of scripts with elevated privileges should be tightly controlled.

Removable media deserves similar treatment. Mission files, log archives, firmware packages, and configuration backups are often transferred through USB storage or other portable devices. These transfers can introduce malicious files, outdated configurations, or data leakage. Organizations should define how removable media is approved, scanned, stored, and reused, with additional controls for workstations that manage sensitive flight operations.

System backups are important for recovery, but they can also contain valuable information for an unauthorized party. A workstation backup may include mission history, network configuration, application preferences, access tokens, and other sensitive records. Backups should be encrypted where appropriate, access should be restricted, and recovery procedures should be tested. A backup is

useful only if it can be restored reliably and without reintroducing obsolete or insecure configuration.

Time synchronization and event logging support both security monitoring and operational troubleshooting. If workstation time is substantially inaccurate, records from the ground control application may not align with logs from the aircraft, telemetry router, or network equipment. Where reliable time services are available, the workstation should use an approved source and record any significant synchronization problems. Logging should capture important application events, configuration changes, authentication failures, and communication interruptions without recording secrets unnecessarily.

Physical access is also relevant. An unattended workstation may expose an active control session, an unlocked mission-planning application, or connected telemetry equipment. The workstation should use appropriate screen locking, device access controls, and secure storage. Ports and interfaces that are not required for normal work should be restricted where feasible. These measures are particularly important when the workstation is used in shared facilities, field locations, or temporary operating sites.

## Protecting Operator Credentials, Mission Files, and Sensitive Flight Data

Operator credentials establish access to the systems and services used to plan, configure, monitor, and manage aircraft operations. Depending on the environment, these credentials may include workstation accounts, remote-access credentials, organizational identity accounts, application passwords, cryptographic keys, or tokens used by supporting services. Their exposure can allow an

unauthorized person to access mission records, alter configuration, or impersonate a legitimate operator.

Credential protection begins with an accurate understanding of which credentials exist and what each one permits. A ground control workstation may have a local user account, while a separate service provides access to shared mission storage. A cloud-connected fleet-management system may use another identity mechanism, and a remote maintenance connection may rely on a separate credential. These identities should not be treated as interchangeable, because compromise of one account may have a different impact from compromise of another.

Each account should receive only the permissions required for its purpose. Operators who plan missions may not need administrative rights over the workstation. Technicians who maintain the operating system may not need unrestricted authority to approve every mission. Personnel responsible for reviewing flight logs may need read-only access to records without the ability to change flight configuration. Separating these responsibilities reduces the consequences of a compromised account and makes accountability clearer.

Strong authentication is important wherever an account can expose sensitive information or influence aircraft operations. Multi-factor authentication can reduce reliance on a password alone when supported by the relevant identity system. Authentication requirements should be proportionate to the risk and designed so that legitimate operators can continue working through approved recovery procedures. Shared accounts should be avoided where individual accountability is required, because they make it difficult to determine who performed a particular action.

Credentials should not be stored in plain-text files, source code, mission notes, or ordinary log messages. Applications that need to retain authentication material should use appropriate operating-

system credential storage or another protected mechanism suitable for the environment. Cryptographic keys used for message signing require special handling because their exposure can undermine the authentication assumptions of the communication system.

Credential reuse can turn a relatively minor incident into a wider compromise. If an operator uses the same password for the ground control workstation, an unrelated online service, and a remote-access account, a breach in one environment may expose the others. Unique credentials, supported by an approved password manager where appropriate, reduce this risk. Organizations should also establish procedures for revoking access when a person changes roles or leaves the organization.

Mission files require protection against unauthorized reading as well as unauthorized modification. A route or inspection plan may reveal where an aircraft will operate and when. In sensitive contexts, this information can expose organizational routines, facility layouts, or the location of valuable assets. Access to mission storage should therefore be restricted according to operational need, with stronger protections for particularly sensitive missions.

Integrity is equally important. A modified mission may remain readable by the ground control application while no longer representing the version approved by the responsible operator. Organizations can reduce this risk through access controls, version history, approval procedures, and integrity verification where supported. When mission files move between systems, the receiving workstation should be able to establish which version is intended for use and whether it has changed since approval.

A mission file should not be trusted merely because it was found in an approved folder or received from a familiar colleague. The file may have been replaced, edited accidentally, or generated using incompatible settings. The workflow should preserve the identity of the source, the relevant version information, and the review status.

Where appropriate, a second person should verify changes that could materially affect safety or mission scope.

Sensitive flight data extends beyond mission files. Telemetry records may contain position histories, operational timing, aircraft identifiers, sensor readings, and information about system faults. Ground control logs may also record operator actions, connection events, and configuration changes. The combined records can reveal a detailed account of an organization's activities, even when each individual record appears harmless.

Data collection should be proportionate to the operational purpose. An organization may need detailed logs for safety investigations or maintenance, but retaining every record indefinitely can create unnecessary exposure. Retention periods should reflect legal requirements, contractual obligations, operational needs, and the value of the information. Records that are no longer required should be securely disposed of according to the organization's policy.

Encryption can help protect sensitive information stored on a workstation, backup device, or approved shared storage system. Where data moves across networks, a suitable protected transport may reduce the risk of interception or modification. These protections should be selected according to the actual architecture. Encryption at rest does not protect data from an attacker who already controls an unlocked workstation, and transport encryption does not prevent an authorized but overly privileged user from accessing information.

Access to flight records should be logged where the sensitivity and operational context justify it. Such records can help determine whether information was accessed or changed unexpectedly. Logging should be configured carefully so that it does not itself expose mission details or credentials. Security records need their own access controls and retention rules, particularly when they contain information about operators and operational activity.

Data sharing requires similar discipline. Sending mission files or flight logs to an external service may improve collaboration or analysis, but it also transfers information outside the immediate workstation. Before enabling such sharing, the organization should understand what data is transmitted, where it is stored, who can access it, and whether the service's retention and deletion practices meet the operational requirements. Unnecessary synchronization should be disabled, especially on systems used for sensitive missions.

Incident response procedures should address suspected credential exposure and mission-data compromise. If an operator account may have been accessed by an unauthorized person, the response should include revoking or resetting the affected credentials, reviewing relevant logs, and determining whether the account was used to alter files or configuration. If a signing key may have been exposed, the organization should assess the scope of systems relying on that key and follow a controlled replacement process. Simply changing a workstation password may not address a compromised protocol key or an unauthorized copy of a mission file.

Recovery should preserve the evidence needed to understand what happened. Before reinstalling software or deleting files, responsible personnel should consider whether relevant logs, configuration records, and affected mission versions need to be preserved. The response should prioritize safe aircraft operation and containment of the incident while maintaining an accurate record of decisions and corrective actions.

# Assessing Ground Control Applications for Configuration and Access-Control

A ground control application should be assessed as both an operational tool and a security-sensitive software system. The purpose of an assessment is not merely to confirm that the application starts, connects to an aircraft, and displays telemetry. It is to determine whether the application behaves according to its intended security policy, whether its configuration is appropriate for its environment, and whether the permissions granted to users and components are limited to legitimate requirements.

An assessment begins with a documented understanding of the system. This includes the application version, operating system, supporting libraries, communication interfaces, expected aircraft types, network connections, user roles, and operational functions. The assessment should distinguish between features enabled by the software and features actually enabled in the deployed configuration. A capability that exists in the application may be disabled, while a seemingly optional service may be active because of a local setting or supporting component.

Configuration review should establish whether the application uses supported software versions and whether unnecessary functionality has been disabled. The review should examine communication settings, local service exposure, file permissions, update mechanisms, logging behavior, and the storage of credentials or sensitive mission data. It should also determine whether the application relies on undocumented assumptions about the network or the trustworthiness of connected endpoints.

Default settings deserve particular scrutiny. Software is often designed to function across a wide range of installations, so default configurations may prioritize compatibility and ease of initial setup. Those defaults may not be appropriate for every operational environment. Unnecessary network listeners, broadly accessible storage locations, permissive access settings, or unused

communication interfaces should be reviewed and restricted where practical.

Access-control testing should establish what each user can view, create, modify, transmit, and administer. The assessment should include normal operator accounts, maintenance accounts, administrative accounts, and any service identities used by supporting components. The goal is to confirm that the privileges assigned to each role correspond to its responsibilities and that the application does not rely solely on interface-level restrictions to enforce important security boundaries.

A hidden or disabled button is not sufficient evidence that an operation is unauthorized. If a user cannot see a command in the graphical interface but can invoke the underlying function through another supported interface, the restriction may be cosmetic rather than effective. Access control should be enforced at the layer that performs the sensitive operation. Testing should verify the actual permissions of the relevant component, not merely the appearance of the interface.

The assessment should examine how the application handles identity changes and multiple connected systems. When an operator selects another aircraft, the interface should update the active target consistently, and commands should be associated with the intended system. If a connection drops and later returns, the application should not retain misleading assumptions about the identity or state of the remote endpoint. These behaviors should be tested in a controlled environment using the documented capabilities of the application.

Communication security should be reviewed alongside access control. The assessor should establish whether the application validates incoming messages, follows the configured authentication policy, and reports failures clearly. If the system uses MAVLink 2 signing, the review should determine whether the intended signing

requirements are actually enforced and whether the ground control application behaves consistently with the autopilot and any intermediate routers. The presence of a signing option in a settings panel is not sufficient proof that all relevant traffic is authenticated.

The review should also consider what happens when a security check fails. An invalid signature, an unsupported message, an unavailable authentication service, or a malformed packet should not cause the application to enter an unsafe state or silently disable its security controls. If the application offers a fallback mode, the assessment should determine whether the fallback is documented, visible to the operator, and appropriate for the intended environment.

Input handling is another major area of review. Mission files, configuration packages, telemetry messages, and log files may originate from external sources. The application should validate these inputs before processing them and should handle invalid content without crashing or performing unintended actions. File import functionality deserves particular attention because an operator may reasonably expect a mission file or log archive to be data rather than executable content. The software should avoid unnecessary execution privileges and should isolate parsing of untrusted files where practical.

The assessment should include the handling of sensitive files. Mission storage should have appropriate access permissions, and temporary files should not expose sensitive content unnecessarily. Export functions should make it clear which information will be included, especially when logs may contain detailed location histories or operational identifiers. If the application automatically synchronizes files or creates backups, the destination and access policy should be understood.

Update mechanisms should be reviewed for authenticity and integrity. The application should obtain software from trusted sources and verify packages where supported. Users should be able

to distinguish a legitimate update from an unverified file presented as an update. Organizations should document the approved installation process and maintain records of deployed versions so that a vulnerable or incompatible release can be identified and corrected.

Logging and auditability should be evaluated against the operational purpose of the application. Important events may include user authentication, changes to mission files, modifications to critical configuration, communication failures, rejected commands, and updates to the software. Logs should provide enough information to support investigation without recording credentials or signing keys. Their integrity and accessibility should also be considered, since an application that records events but permits unrestricted alteration of the records provides limited investigative assurance.

Usability should form part of the assessment because security controls can fail through confusing design. Warnings should identify the condition that requires attention and communicate its operational significance. The interface should distinguish a loss of telemetry from a confirmed aircraft fault, and it should distinguish a command that has been sent from an action that has been completed. Excessive warnings can cause operators to ignore important alerts, while insufficient warnings can hide meaningful changes in system state.

Assessment results should be based on reproducible observations. Each finding should describe the affected component, the conditions required to observe the behavior, the expected result, the actual result, and the potential operational consequence. A suspected weakness should be labeled as such until the available evidence supports a stronger conclusion. The assessor should also state the limits of the evaluation, including any aircraft hardware, software versions, or communication paths that were not tested.

Corrective actions should be prioritized according to both likelihood and impact. A configuration weakness that exposes a command interface to an unnecessary network may require prompt attention. A minor display inconsistency may be less urgent unless it can mislead an operator during a safety-critical event. Risk assessment should consider the aircraft's role, the accessibility of the affected component, the sensitivity of the data, and the practical consequences of exploitation or accidental misuse.

After remediation, the relevant tests should be repeated to confirm that the issue has been resolved. The review should also check that the correction has not impaired necessary functions, such as legitimate telemetry reception, mission upload, or emergency procedures. Security assurance is strongest when configuration, access-control policy, application behavior, and operational procedures are evaluated together and maintained as the system evolves.

# Drone Networking and Connected Services

## IP Networking Fundamentals for Connected Unmanned Aircraft

Modern unmanned aircraft increasingly depend on networked systems to exchange flight information, support remote operation, coordinate onboard computing resources, and connect with external applications. Even an aircraft designed primarily for autonomous flight may rely on network communication for mission preparation, telemetry monitoring, software maintenance, payload management, or transfer of recorded data. Understanding these connections is essential to evaluating the security of an unmanned aircraft system because the network can extend the aircraft's operational environment far beyond its physical structure.

Internet Protocol networking provides a common foundation for communication between computers and embedded devices. Instead of requiring every application to understand the electrical characteristics of a physical connection, IP networking allows software to exchange packets through a standardized addressing and routing system. This makes it possible for a companion computer, ground control workstation, payload processor, and remote service to communicate even when they operate on different hardware platforms.

An IP network carries information in packets that include addressing and control information alongside the data being transported. The

source address identifies the address associated with the sending interface, while the destination address identifies the intended recipient. Routers use addressing and routing information to determine how packets should move between networks. Within a local network, switches and other networking equipment help deliver traffic between connected devices.

These mechanisms support flexible system designs, but they do not inherently establish that a device or message is trustworthy. An IP address is a network identifier, not proof of identity. A packet arriving from an expected address may still have been generated by a compromised device, and an endpoint connected to the same local network may not be authorized to access every service available on that network. Security requires additional controls that establish identity, restrict permissions, and validate the information being exchanged.

A connected unmanned aircraft system may contain several logically distinct networks. One may support communication between the flight controller and a companion computer. Another may connect the aircraft to a ground control workstation. A separate network may carry payload imagery or data from an onboard sensor. Remote maintenance services may use yet another connection. The boundaries between these networks should reflect their different purposes and risk levels.

The flight controller often operates under strict timing and reliability requirements. Its primary responsibility is to manage aircraft behavior using sensor inputs and control algorithms. A companion computer may provide higher-level functions such as image processing, object detection, mission coordination, or communication with external services. These systems can exchange data without necessarily sharing the same security requirements. The design should prevent a compromise of a less trusted application from automatically granting unrestricted access to the flight-control functions.

Network protocols operate at different layers, each addressing a particular aspect of communication. IP provides addressing and routing, while transport protocols such as Transmission Control Protocol and User Datagram Protocol support communication between applications. Transmission Control Protocol establishes a connection with ordered delivery and retransmission mechanisms. User Datagram Protocol provides datagram delivery without the same built-in guarantees of ordering or retransmission. The appropriate choice depends on the application's requirements, including latency, reliability, bandwidth, and tolerance for packet loss.

For example, a service transferring a mission file may benefit from reliable, ordered delivery because missing or altered portions of the file can invalidate the transfer. A telemetry stream may prioritize timely updates, with the application deciding how to handle occasional missing packets. Neither transport choice automatically supplies application-level authorization. A reliable connection can carry an unauthorized request just as effectively as an authorized one if the receiving service does not enforce appropriate access controls.

Address allocation is another important part of network design. Some systems use statically assigned addresses, while others obtain addresses through dynamic configuration services. Static addressing can make critical interfaces predictable, but it requires careful management to prevent conflicts and outdated records. Dynamic allocation simplifies administration in some environments but may introduce uncertainty when a service expects an endpoint to remain at a particular address. Regardless of the method, the network inventory should identify the interfaces that matter to aircraft operation and the services permitted to use them.

Subnetting divides an address space into smaller network segments. In an unmanned aircraft environment, this can help separate ground control devices from general office computers or isolate payload

processing from administrative services. Routing rules determine which segments can communicate. A well-designed arrangement limits unnecessary paths and makes it easier to monitor traffic between systems with different responsibilities.

However, separate IP subnets do not automatically constitute strong security boundaries. If a router allows unrestricted traffic between them, or if a workstation has interfaces connected to both networks, the separation may offer little practical protection. Network architecture must therefore be assessed through actual routing rules, firewall policies, interface configurations, and the permissions enforced by the applications themselves.

Connected aircraft systems also need to account for address translation and intermediary services. A network may use private addresses internally while a gateway manages communication with an external network. A companion computer may relay information between a local serial interface and an IP connection. These arrangements can simplify connectivity, but each gateway or bridge introduces another point where traffic may be forwarded, filtered, inspected, or misdirected.

The presence of an intermediary should not be mistaken for security. A gateway that forwards all traffic without enforcing meaningful restrictions can extend the reach of an untrusted endpoint into a more sensitive network. Where possible, gateway rules should allow only the communication required for the defined operational purpose. Services should not be exposed simply because forwarding them is convenient.

Network availability is particularly significant for aircraft that depend on connected services. A loss of communication may prevent an operator from receiving current telemetry, transferring a mission, or accessing a remote maintenance function. The aircraft should therefore have a clearly defined response to the loss of each network dependency. Essential flight functions should not depend

unnecessarily on a remote service that may become unavailable during operation.

The degree of independence required depends on the aircraft's intended role. A platform designed to complete a locally stored mission may continue safely without a persistent external connection, subject to its configured failsafes and operating limitations. A remotely managed platform may require regular connectivity for certain functions. In either case, the system design should identify which functions can continue without the network and which require a controlled transition when communication fails.

## Wi-Fi, Ethernet, and Local Communication Interfaces

Wi-Fi and Ethernet provide common methods for connecting computers and embedded devices within an unmanned aircraft system. Ethernet is frequently used where a wired connection is practical, while Wi-Fi supports wireless connectivity between mobile or distributed components. Both can carry IP traffic and support many of the same application protocols, but their physical characteristics and exposure patterns differ.

Ethernet provides a wired link through network interfaces and suitable cabling. In a laboratory, workshop, or fixed ground station, it can offer predictable connectivity and relatively stable performance. On an aircraft, Ethernet may connect a companion computer to a payload processor, a network-capable camera, or other equipment that exchanges substantial amounts of data. Wired communication can reduce exposure to nearby wireless participants, but physical access to a network port may still create a security concern.

A port that is available for maintenance or debugging can become an entry point if it connects directly to a sensitive network. Physical access controls, interface restrictions, and appropriate network policies should therefore be considered alongside the security of the software using the port. Where a connector is not required for normal operation, its exposure should be evaluated and controlled according to the aircraft's maintenance and operational needs.

Wi-Fi provides flexibility when cables are impractical or when an operator needs to move freely around an aircraft or test area. It may be used for telemetry, local configuration, payload data, or access to a companion computer. Its convenience also creates exposure to devices within radio range, depending on the network's configuration and the environment. The actual risk depends on authentication settings, encryption, network reachability, and the permissions of services available after connection.

Wireless network security begins with appropriate authentication and modern encryption supported by the equipment. Weak passwords, shared credentials used across multiple devices, and obsolete security modes can undermine protection. Credentials should be unique where practical, and access should be restricted to authorized users and devices. When the equipment supports stronger enterprise authentication, the suitability of that approach should be considered in relation to the operational environment and maintenance capabilities.

A secured Wi-Fi connection does not make every application on the network secure. Once a device has joined the network, it may still be able to reach services that were never intended for it. Network-level authentication should therefore be combined with host firewalls, service-specific authentication, and restrictions on communication between segments. A visitor or maintenance device should not automatically receive access to flight-control interfaces merely

because it has joined a wireless network used for legitimate operations.

The network name and other visible configuration details should not be treated as secrets. Concealing a network identifier does not provide meaningful authentication, and a determined observer may still identify the network through normal wireless operation. Security should rely on proper authentication, encryption, access control, and sensible network design rather than obscurity.

Local interfaces may include serial ports, USB connections, Ethernet ports, wireless adapters, and internal communication buses. Although some of these interfaces do not carry IP traffic directly, they may connect to components that do. A USB network adapter, for example, can create a new IP interface, while a companion computer may bridge traffic between an Ethernet connection and a serial communication channel used by an autopilot.

Bridging and forwarding behavior deserves special attention because it can unintentionally connect networks that were intended to remain separate. A companion computer might be configured to pass telemetry between a flight controller and a ground control station, but an overly broad forwarding rule could also expose additional services. The security review should establish which interfaces are connected, which traffic is forwarded, and whether the computer can be used as a route between otherwise isolated systems.

Network discovery and automatic connection features can create further uncertainty. A workstation may connect automatically to a previously used wireless network, or a service may advertise its availability to nearby devices. Such convenience features should be assessed in the context of the operating environment. A ground control workstation should not silently connect to an untrusted network and then expose sensitive services through that connection.

Local communication interfaces also vary in their susceptibility to physical interference, environmental damage, and accidental disconnection. A security design must account for reliability as well as unauthorized access. Connectors should be secured appropriately, cables should be routed to reduce accidental damage, and the system should report interface failures clearly. A link that fails intermittently can create conditions in which operators misinterpret incomplete data or assume that a command has been delivered.

The selection of Wi-Fi or Ethernet should reflect operational needs rather than a blanket assumption that one is always more secure. Ethernet can reduce wireless exposure but may still be vulnerable through accessible ports or compromised connected equipment. Wi-Fi can be configured securely but requires careful management of credentials, radio coverage, and client access. In both cases, the strongest results come from minimizing unnecessary connectivity and enforcing explicit boundaries between operational functions.

## Network Discovery, Service Exposure, and Access-Control Boundaries

Network discovery helps administrators identify which devices are present, which addresses they use, and which services they provide. In a connected aircraft environment, this information supports asset inventory, troubleshooting, configuration review, and security assessment. It also helps reveal unexpected connections that may have been introduced through maintenance, software installation, or changes to the network.

A device can expose several services even when its primary role appears simple. A companion computer may provide telemetry forwarding, a diagnostic interface, remote administration, a file-

transfer service, and an application programming interface. A ground control workstation may expose file sharing, remote access, or local services used by supporting software. Each service increases the number of ways another device can interact with the system.

Service exposure should be evaluated according to necessity. A service required for normal operation should be documented, configured securely, and limited to the appropriate users or network segments. A service used only during maintenance should be disabled or made inaccessible during ordinary operation where practical. A service that is no longer required should be removed rather than left active because its purpose is uncertain.

The presence of an open network port does not automatically indicate a vulnerability. Some services are necessary for legitimate communication, and an open port alone does not establish that the service can be exploited. The relevant questions are whether the service is expected, whether it is accessible to the intended clients, whether it requires suitable authentication, whether it is running a supported implementation, and whether its permissions are appropriate.

An asset inventory should record the purpose of each network-connected device and the services it is expected to expose. For an aircraft system, this might include the flight-control interface, telemetry router, companion computer, payload processor, ground control workstation, and remote management service. The inventory should also identify which services are necessary during flight, which are used only for maintenance, and which should never be reachable from an external network.

Service discovery can be performed in an authorized test environment using approved inventory tools and configuration review. Passive methods, such as inspecting device documentation, firewall rules, service configuration, and network logs, can reveal much of the required information without generating additional

traffic. Active testing may be useful when the scope and permission are clear, but it should be conducted carefully to avoid disrupting time-sensitive systems.

The distinction between a listening service and an accessible service is important. A process may listen on a local interface but remain unreachable from other devices because of firewall rules. Conversely, a service that appears intended for local use may become accessible externally if it binds to all interfaces or if routing rules expose it. Assessment should therefore examine both the service configuration and the actual network paths available to potential clients.

Access-control boundaries determine which users, applications, and devices can reach particular resources. These boundaries should correspond to the system's operational responsibilities. A telemetry consumer may require permission to receive status messages but not to change mission parameters. A maintenance workstation may need access to a diagnostic service when the aircraft is disarmed but should not automatically retain the same access during normal flight operations.

A common design weakness occurs when network reachability is treated as sufficient authorization. If a device can connect to a service, the service may assume that the device is trusted. This assumption is unsafe in environments where several devices share a local network or where network segments connect through gateways. Services should authenticate clients and enforce permissions appropriate to the requested operation.

The principle of least privilege applies to both users and services. A network daemon should run with only the permissions required for its task. An application programming interface should expose only the operations necessary for its intended clients. A file-transfer service should not provide unrestricted access to unrelated system directories. If a component is compromised, limited privileges can reduce the amount of damage it can cause.

Service interfaces also require careful input validation. A network service may receive requests from software versions it was not designed to support, malformed data, or repeated requests that consume excessive resources. It should validate message formats, restrict input sizes, handle errors safely, and avoid exposing sensitive diagnostic details to unauthorized clients. Logging should support investigation without recording passwords, keys, or unnecessary operational information.

Discovery information itself may be sensitive. A detailed inventory can reveal device names, software versions, network structure, service roles, and operational relationships. Such information helps defenders manage a system, but it can also assist an unauthorized party in understanding its architecture. Inventory records and detailed network diagrams should therefore be stored and shared according to the sensitivity of the operation.

Changes to the network should trigger a review of service exposure. Installing a new payload computer, adding a wireless bridge, replacing a router, or enabling remote maintenance can change the reachable services without modifying the aircraft's primary flight software. Configuration drift should be detected through comparison with the approved network baseline. Unexpected services or new communication paths should be investigated rather than assumed to be harmless.

## Secure Remote Administration and Network Segmentation

Remote administration allows authorized personnel to maintain equipment, inspect logs, update software, and diagnose faults

without being physically present at the device. For distributed unmanned aircraft operations, this capability can reduce maintenance time and improve access to technical expertise. It also introduces a path into the system that must be protected with appropriate identity verification, permissions, monitoring, and operational restrictions.

Remote access should be enabled only when there is a defined requirement. A system that does not need remote administration should not expose a remote management service merely because the software provides one. Where remote access is necessary, the organization should identify which users may connect, which devices they may administer, and which operations they are permitted to perform.

Authentication should be strong enough for the risk associated with the remote interface. Multi-factor authentication can reduce the likelihood that a stolen password alone will provide access. Where supported, individual accounts and cryptographic authentication can improve accountability and reduce dependence on shared secrets. Credentials should be revoked promptly when access is no longer required or when compromise is suspected.

Remote access should also be restricted by network policy. A management service intended for a trusted administrative network should not be exposed indiscriminately to every interface. Firewalls and routing rules can limit access to approved management endpoints. A protected access gateway may provide an additional control point, provided that the gateway itself is securely configured, maintained, and monitored.

Administrative access should be separated from ordinary operational traffic where practical. Remote maintenance may require broad permissions over a companion computer, but those permissions should not automatically grant direct control over the flight controller. The system should maintain a clear distinction between

administering the supporting computer and issuing commands that affect aircraft behavior.

Network segmentation is a practical method for enforcing this separation. It divides a network into smaller zones according to purpose and trust level, then controls communication between those zones. A connected aircraft system might separate flight-related interfaces, payload processing, operator workstations, maintenance equipment, and external services. The exact boundaries should be based on the architecture and the consequences of compromise rather than copied from a generic network template.

Segmentation is most effective when each zone has a clear purpose and explicit communication requirements. If a payload processor needs to send sensor data to a ground station, the network policy can permit that flow without allowing the payload processor to initiate unrelated connections to the flight-control workstation. If an administrative computer needs to update a companion computer, the rule can be restricted to the required management service rather than permitting unrestricted access to the entire aircraft network.

A default-deny approach to inter-zone traffic can reduce unnecessary connectivity. Under this model, communication is blocked unless it has been explicitly permitted. The policy should be implemented carefully, however, because legitimate operational dependencies may not be obvious at first. Testing and documentation are necessary to ensure that segmentation does not interfere with required telemetry, safety functions, or maintenance procedures.

A workstation connected to two network segments can undermine separation if it forwards traffic between them. This can happen intentionally, through a routing configuration, or unintentionally, through a software bridge or internet-sharing feature. Systems that connect operational equipment to an administrative network should therefore be reviewed for forwarding behavior, interface binding, and host firewall rules.

Remote administration should not depend on an assumption that the internal network is automatically trustworthy. An internal device may be compromised, misconfigured, or connected through an unauthorized path. Services should authenticate users and enforce permissions even when requests originate from a network considered trusted. This layered approach reduces the consequences of a single network-control failure.

Maintenance sessions should be limited to the time and scope required. If an aircraft is normally operated without remote administrative access, the service can be enabled through a controlled procedure when maintenance is necessary and disabled afterward. The process should record who authorized the access, which system was administered, what changes were made, and whether the system was returned to its approved configuration.

Remote changes to operational parameters deserve particular caution. A technician may need to modify a setting to diagnose a fault, but the change could affect later flight behavior. The system should distinguish diagnostic activity from operational approval and preserve a record of any configuration change that could influence safety. Where appropriate, a second person should review consequential changes before the aircraft returns to service.

The organization should also plan for remote-access failure. Authentication services may become unavailable, management links may be interrupted, or a remote session may end while a configuration change is in progress. The system should avoid leaving critical services in a partially configured state. Recovery procedures should define how to restore a known-good configuration and how to regain authorized access without introducing an uncontrolled alternative connection.

Monitoring supports the security of remote administration. Relevant records may include successful and failed logins, session duration, changes to access policy, service restarts, configuration modifications, and attempted access to restricted endpoints. Alerts should focus on meaningful events, such as repeated authentication failures or unexpected management access during an operational period. Monitoring should be proportionate so that routine activity does not overwhelm the people responsible for responding.

Segmentation policies must be maintained as the aircraft system changes. Adding a new camera, replacing a companion computer, or introducing a remote analytics service may create new communication requirements. These changes should be documented and tested before deployment. A network that was well segmented at installation can gradually become permissive if temporary exceptions are never removed or if new equipment is connected without review.

## Assessing Network Configurations Within an Authorized Test Environment

An authorized network assessment establishes whether the actual configuration of an unmanned aircraft system matches its intended security design. The objective is to identify unnecessary exposure, weak access controls, incorrect routing, insecure services, and conditions that could interfere with reliable operation. The assessment should be planned to avoid disrupting operational aircraft or systems outside the approved scope.

The first requirement is a clearly defined test environment. This may consist of a simulated aircraft network, development hardware, a laboratory companion computer, a test ground control workstation, or a representative network assembled from equipment under the

assessor's control. The scope should identify the devices, interfaces, address ranges, services, and testing methods that are authorized. Production networks and third-party systems should be excluded unless explicit permission covers them.

Before active testing begins, the assessor should document the intended network architecture. The diagram should show which devices communicate, which interfaces connect the network segments, and which services are expected to be available. The assessment should also record the intended access rules and identify the functions that must remain available throughout the test. This baseline provides a reference for distinguishing expected behavior from unexpected exposure.

Configuration review is often the safest starting point. Router settings, firewall policies, interface bindings, service configuration, system logs, and device inventories can reveal whether the intended security boundaries are actually implemented. Reviewing these records can also identify discrepancies between documented architecture and deployed configuration before active probes are introduced.

The next stage establishes which devices and services are present within the authorized scope. Approved discovery methods can be used to compare observed endpoints with the asset inventory. The assessor should verify the identity and role of each discovered device rather than assuming that a hostname or address proves its purpose. Unexpected devices should be documented and investigated, particularly if they appear on a network intended for flight-related communication.

Service review should determine whether each exposed service is necessary, properly configured, and restricted to appropriate clients. The assessor should distinguish between services that are intentionally available and those that appear because of default settings or unnecessary supporting software. A service that is not

required for operation should generally be disabled or isolated, subject to the platform's documented requirements.

Access-control testing should verify the difference between intended and actual permissions. For example, a read-only monitoring workstation should not be able to perform administrative functions merely because it can reach a companion computer. A maintenance service should not accept unauthenticated requests if the approved design requires authenticated access. Tests should use authorized accounts and controlled inputs, with care taken not to alter operational settings unintentionally.

Firewall and segmentation testing should establish whether traffic follows the documented policy. The assessor can verify that permitted communication works and that prohibited paths remain blocked. This should include review of any device connected to multiple network segments, since such a device may create a route around otherwise effective restrictions. Any exceptions should have a documented operational reason and an assigned owner responsible for reviewing them.

The test should also evaluate failure behavior. In a laboratory environment, the assessor can examine what happens when a network interface becomes unavailable, a service restarts, or a connection is interrupted. The goal is to determine whether the system recovers predictably and whether a failure causes an unintended expansion of access. A secure configuration should not silently disable restrictions or expose a management service simply because another component has failed.

Network reliability should be evaluated alongside confidentiality and access control. Excessive traffic, poorly configured routing, or a malfunctioning service can affect latency and availability. Tests should remain within safe limits and avoid creating conditions that could damage hardware or disrupt unrelated systems. For time-sensitive components, performance observations should include the

effects of normal traffic and controlled stress conditions on message delivery and application responsiveness.

If wireless interfaces are in scope, the assessment should review authentication settings, encryption configuration, permitted clients, network reachability, and the behavior of devices when the preferred network becomes unavailable. The evaluation should be conducted within the approved environment and should avoid interfering with neighboring wireless systems. The purpose is to confirm that only intended devices can connect and that joining the network does not grant unnecessary access to other services.

Configuration drift should be checked by comparing the observed state with the approved baseline. Differences may include new listening services, altered firewall rules, changed network addresses, additional wireless interfaces, or remote-access features that were not present during the original review. Each difference should be classified according to whether it is authorized, necessary, and adequately protected.

The assessor should pay attention to temporary exceptions. During development or maintenance, teams may open a service to simplify troubleshooting or allow broader access to complete a task. If the temporary setting remains in place, it can become a permanent weakness. The review should identify such exceptions and determine whether they have been removed or formally approved with appropriate restrictions.

Results should be documented in a form that supports corrective action. A useful finding describes the affected device or service, the observed configuration, the expected policy, the evidence supporting the finding, and the potential consequence. It should distinguish between a confirmed exposure and a hypothetical concern. For example, an unnecessary service reachable from a general-purpose network presents a different risk from a service that listens locally

but cannot be reached through the network's actual routing and firewall rules.

The severity of a finding depends on context. An exposed administrative service on a system connected directly to a sensitive aircraft network may present a greater concern than an informational service on an isolated test workstation. Risk assessment should consider reachability, authentication, privilege level, the sensitivity of accessible data, the potential effect on flight operations, and the availability of compensating controls.

Remediation should address the underlying cause rather than only the immediate observation. If a service is unnecessarily exposed, the appropriate response may be to disable it, restrict its interface binding, or enforce a firewall rule. If several network segments communicate too freely, the organization may need to revise its routing policy and define explicit allowed flows. If a device has excessive privileges, service permissions and administrative roles should be reviewed.

After corrective changes, the relevant tests should be repeated to confirm that the intended restrictions are effective. The assessor should also verify that required communication remains functional. A firewall rule that blocks an unauthorized management path but also prevents essential telemetry from reaching the operator may create a different operational problem. The objective is a configuration that provides meaningful protection while preserving the functions needed for safe and reliable aircraft operation.

The assessment record should identify the environment tested, the equipment and software versions, the scope limitations, the findings, and the corrective actions. It should state which network paths and services were examined and which were outside the assessment. This record supports future maintenance, makes changes easier to review, and provides a defensible basis for determining whether the network continues to meet its intended security requirements.

# Cellular Networks and Remote Drone Connectivity

## Cellular Modems, Subscriber Identity, and Mobile Network Integration

Cellular connectivity has expanded the operating capabilities of unmanned aircraft by allowing them to exchange information across distances that exceed the practical reach of many conventional radio links. Through mobile networks, an aircraft can transmit telemetry to a remote operations center, send sensor data to an authorized processing service, receive mission updates, or support maintenance activities without requiring a dedicated point-to-point communication link for every connection. These capabilities are valuable for infrastructure inspection, environmental monitoring, agricultural operations, surveying, and other applications in which aircraft and operators may be separated by substantial distances.

The integration of cellular communication introduces a distinct set of security considerations. An aircraft that previously relied on a local telemetry connection may become dependent on mobile network infrastructure, subscriber credentials, internet routing, cloud services, and remote management platforms. Each additional dependency can improve operational flexibility, but it also creates another component that must be configured, maintained, and protected. Cellular connectivity should therefore be understood as an extension of the aircraft's communication architecture rather than as a self-contained security solution.

A cellular modem provides the interface between the aircraft's onboard computing environment and a mobile network. Depending on the design, the modem may be integrated into a companion computer, installed as a separate module, or connected through an interface such as USB, a serial connection, or an embedded hardware bus. It manages the radio communication required to access the mobile network and exposes a data interface through which onboard software can establish network connections.

The modem does not necessarily control the aircraft directly. In many designs, a flight controller handles stabilization, navigation, and immediate flight-control responsibilities, while a companion computer manages higher-level connectivity. The companion computer may receive telemetry from the flight controller, package information for transmission, and forward selected data to a remote service. Commands received through the network may follow a reverse path, passing through application software before reaching the flight controller, where additional validation and authorization should take place.

This separation is important because cellular connectivity can fail independently of the aircraft's basic flight-control functions. Coverage may be lost, a modem may restart, a network session may expire, or the mobile operator may experience an outage. A well-designed aircraft should have clearly defined behavior for each condition. Functions essential to immediate flight safety should not depend unnecessarily on the uninterrupted availability of a remote network.

Mobile networks use subscriber identity and authentication mechanisms to determine whether a device is permitted to access network services. Depending on the technology and implementation, subscriber information may be stored on a removable subscriber identity module or an embedded subscriber identity component. These mechanisms support the relationship between a device, its

subscription, and the mobile operator's authentication infrastructure.

A subscriber identity module is not simply a password stored inside the aircraft. It participates in a standardized authentication process that uses subscriber-specific credentials and network procedures. The mobile network verifies the subscription according to its authentication architecture, and the device establishes access subject to the operator's policies. This process controls access to the cellular service, but it does not automatically authenticate the aircraft's application software or authorize commands sent to the flight controller.

That distinction is fundamental. A modem may be legitimately registered with a mobile network while the application using its connection is vulnerable to unauthorized access. Likewise, a valid subscriber identity does not prove that a remote server sending a command is authorized to control the aircraft. Cellular subscription authentication, application authentication, and aircraft command authorization must be treated as separate security layers.

Subscriber credentials should be managed as sensitive operational assets. A removable module may be physically extracted from an accessible device, while an embedded identity component may depend on the security of the hardware and its provisioning process. Organizations should understand how subscriptions are activated, who can modify them, how credentials are recovered or replaced, and how access is revoked when a device is lost, retired, or transferred to another operator.

The cellular modem itself requires careful configuration. Its firmware, supported network technologies, management interfaces, and exposed services can influence the security of the wider system. A modem may offer diagnostic commands, configuration utilities, status interfaces, and software-update mechanisms. These functions should be restricted to the components and personnel that require

them. Unnecessary management interfaces should not be exposed through general-purpose network connections.

Modem firmware is part of the aircraft's software supply chain. Updates may correct vulnerabilities, improve compatibility, or change network behavior. The organization should obtain updates through trusted channels, verify authenticity where supported, and test them against the aircraft's operational requirements. A modem update that changes connection handling or interface behavior may affect applications that depend on its existing configuration, so deployment should be controlled rather than treated as an isolated maintenance task.

Mobile network integration also depends on the subscription and service plan. Some subscriptions provide ordinary internet connectivity, while others support private addressing, managed connectivity, fixed addressing, or other enterprise features. The exact capabilities vary by operator and service. A private cellular network configuration may reduce certain forms of exposure, but its security still depends on access controls, routing policy, device configuration, and the protection of the applications that communicate over it.

An aircraft may use more than one cellular operator to improve coverage or resilience. Such designs can provide an alternative connection when one network becomes unavailable, but they also increase configuration complexity. Each subscription, modem profile, network route, and failover mechanism must be managed consistently. A secondary connection should not become a less protected path into the aircraft merely because it was introduced to improve availability.

The physical environment can also affect connectivity. Antenna placement, shielding, aircraft structure, interference, and movement can influence radio performance. A connection that performs reliably during ground testing may behave differently during flight or in a remote operating area. Network design should account for expected

coverage, antenna requirements, modem capabilities, and the consequences of temporary signal degradation.

A sound integration process begins with a clear definition of the modem's role. If it is intended only to transmit telemetry, the architecture should avoid granting it unnecessary access to command functions. If it must carry authorized remote commands, the receiving application should authenticate the command source, validate each request, and enforce operational permissions independently of the cellular connection. Limiting the modem's access to the minimum required network paths reduces the consequences of a compromise elsewhere in the system.

## Remote Telemetry, Command Transport, and Data Connectivity

Remote telemetry allows an operator to observe an aircraft from a location that may be far beyond the range of a local radio connection. Depending on the platform, telemetry may include position, altitude, speed, attitude, battery condition, flight mode, sensor status, mission progress, and system warnings. Cellular connectivity can deliver this information to a remote ground control station or a service that aggregates data from several aircraft.

The value of remote telemetry depends on the quality and timeliness of the information received. A network connection may remain established while experiencing significant latency, packet loss, or interruptions. The ground control application should therefore distinguish between a functioning network session and telemetry that is sufficiently current to support operational decisions. A connected modem does not guarantee that the aircraft's latest status has reached the operator.

Telemetry applications should track freshness and report communication problems clearly. When updates stop arriving, the interface should not continue presenting old values as current. It should identify the last known update or otherwise communicate that the information may be stale. Where operationally appropriate, the system can also distinguish between loss of cellular connectivity, loss of communication with the flight controller, and a failure of the remote application.

These conditions have different causes and consequences. A modem may retain network registration while the companion computer has stopped forwarding telemetry. The companion computer may function correctly while the mobile network is unavailable. A remote service may receive packets but fail to update the operator's display. Monitoring should identify which stage of the communication path has failed rather than reducing every problem to a generic disconnected status.

Remote command transport requires stronger controls than telemetry reception alone. A command may request a change in flight mode, alter mission parameters, initiate an operation, or modify an onboard configuration. If commands are carried over a cellular connection, the system must protect the path from the remote operator to the component that ultimately evaluates and executes the request.

A typical architecture may include the ground control application, a remote communication service, a cellular data connection, an onboard companion computer, and the flight controller. Each component has a different responsibility. The ground control application creates an authorized request, the remote service routes or manages communication, the cellular network transports data, and the onboard software evaluates the request before forwarding any permitted operation. Security should be enforced at the relevant

decision points rather than assumed from the presence of the cellular connection.

A remote service should not automatically trust a request simply because it arrives from an address associated with a known mobile network. Network location is not a substitute for authentication. The service should establish the identity of the requesting user or application, verify the request's integrity, and determine whether the requested action is permitted. The onboard system should independently enforce the command policy that applies to the aircraft.

This layered approach helps prevent a compromise in one component from granting unrestricted control over the entire system. If a remote dashboard account is compromised, strong command authorization may limit what the attacker can do. If a routing service is misconfigured, onboard validation may still prevent an unauthorized request from changing flight behavior. No single safeguard eliminates every risk, but independent controls reduce the likelihood that one failure becomes a complete compromise.

The design should distinguish between data transport and command semantics. A network service may successfully deliver a message without understanding whether the requested action is safe or appropriate. The aircraft-side application should validate the command against the current operational state and the aircraft's supported functions. For example, a request that is technically valid may still be unacceptable if required preconditions have not been met.

Command acknowledgement also requires careful interpretation. A remote service may confirm receipt of a request, but that does not prove the aircraft accepted it. The onboard application may reject the command, or the operation may remain incomplete because the aircraft cannot satisfy the necessary conditions. The remote interface

should report these states separately where the system supports them.

Cellular networks introduce variable latency that can complicate command sequencing. A command may be delayed while a later status message arrives through a different processing path. The system should not assume that network delivery order always corresponds to operational order. Commands that depend on the current state should be evaluated when they reach the relevant control component, and stale requests should be rejected or handled according to a documented policy.

Retries require similar discipline. An application may resend a request after an acknowledgement fails to arrive. If the operation is not designed to tolerate repetition, the second request could produce an unintended result. The application should distinguish between operations that are safe to repeat and operations that require unique transaction identifiers, explicit state checks, or confirmation before another attempt.

Data connectivity may also support payload uploads, imagery transmission, sensor analytics, and transfer of flight records. These uses can generate considerably more traffic than basic telemetry. A platform that sends high-resolution imagery may compete for bandwidth with time-sensitive status information, particularly when coverage is weak or network capacity is shared. The architecture should establish appropriate priorities and avoid allowing large data transfers to undermine essential communications.

Traffic management should reflect operational importance. Routine logs and nonurgent imagery can often tolerate delays, while selected telemetry messages may need more consistent delivery. The precise policy depends on the mission and the available network. Systems should be tested under representative load conditions to determine whether congestion affects important updates or causes excessive resource consumption on the companion computer.

Bandwidth limitations should also influence the choice of data sent remotely. Continuous transmission of every available sensor stream may be unnecessary for many missions. A more efficient design sends the information required for live operation and retains other data locally for later transfer. This reduces network usage, limits unnecessary exposure of sensitive information, and may improve resilience in areas with intermittent coverage.

Local storage and buffering require careful handling. When connectivity is unavailable, the companion computer may retain telemetry or sensor data until the network returns. The system should define storage limits, retention periods, and recovery behavior. It should avoid exhausting disk space during an extended outage and should not automatically transmit stale commands merely because they were queued before the interruption.

Remote telemetry should also be treated as operationally sensitive. Position histories, mission routes, flight timing, and equipment status may reveal patterns that are valuable to an unauthorized observer. The organization should decide which data must be transmitted, how long remote services may retain it, and which users are permitted to view it. Data minimization is especially useful where the remote platform does not require every detail produced by the aircraft.

# Authentication and Encryption Across Cellular Communication Paths

Cellular communication relies on several security mechanisms, but no single mechanism protects every stage of the connection. Mobile network authentication governs access to the cellular service.

Network-layer and transport-layer controls may protect data while it travels between endpoints. Application authentication determines which users or services may interact with a remote platform. Command authorization determines whether a particular request may influence the aircraft. These functions complement one another and should not be treated as interchangeable.

The cellular network's subscriber authentication process helps prevent an unauthorized device from simply using another subscriber's credentials without the required authentication material. It does not necessarily protect application data from every threat between the aircraft and the destination service. Once traffic leaves the mobile operator's managed network, it may traverse additional networks and infrastructure. The security of that path depends on the protections applied to the data and the endpoints.

Encryption protects information from being read by parties that do not possess the required decryption material, subject to the properties and limitations of the encryption mechanism. For remote drone connectivity, suitable encryption may protect telemetry, mission files, operator credentials, and commands as they move between the aircraft and authorized services. The exact design depends on the network architecture, application protocols, and security requirements.

Encryption in transit should use current, supported protocols and correctly validated certificates or other appropriate authentication material. Merely enabling encryption is insufficient if the client accepts invalid certificates, ignores identity mismatches, or permits insecure fallback behavior. The system must verify that it is communicating with the intended endpoint, not simply that some form of encrypted session has been established.

Certificate validation is particularly important for remote management and cloud-connected systems. If an application fails to verify the identity of the remote service, an intermediary may be able

to impersonate that service under certain conditions. The application should validate the service identity according to the chosen protocol and should handle expired, invalid, or mismatched certificates in a controlled manner. Security warnings should not be designed so that users routinely bypass them to restore connectivity.

Mutual authentication may be appropriate for sensitive connections. In a mutual-authentication design, both the client and server establish their identities rather than requiring the client to trust the server alone. This can help restrict access to a remote service to registered devices or authorized applications. Its effectiveness depends on secure credential provisioning, protected storage, certificate or key lifecycle management, and the ability to revoke access when a device is compromised.

Application credentials should be distinct from cellular subscriber credentials. A mobile subscription should not be used as the sole basis for deciding whether an application may issue commands. Likewise, a password that permits access to a remote dashboard should not automatically grant every command capability available to the aircraft. Role-based permissions and command-specific authorization can limit the impact of stolen or misused credentials.

Where the system uses message signing, the signing mechanism should be configured according to the protocol's supported features and the security requirements of the application. Message signing can help authenticate message origin and detect unauthorized modification, but it does not inherently encrypt the message contents. A signed command may still be readable by an observer unless confidentiality is provided separately. Conversely, encryption does not automatically guarantee that a user is authorized to perform the requested operation.

Key management is therefore a central concern. Cryptographic keys must be generated, distributed, stored, rotated, and revoked through a controlled process. A shared secret installed on every aircraft may

simplify deployment, but it can increase the impact of a single device compromise. Where practical, credentials should be scoped to individual devices or clearly defined groups so that one compromised unit does not automatically undermine every aircraft in the fleet.

Keys and credentials should not be embedded in source code, stored in ordinary text files, or exposed through diagnostic logs. The system should use protected storage appropriate to the hardware and operating environment. If a companion computer is accessible to maintenance personnel, the organization should assess whether its credentials can be extracted through local files, debugging interfaces, backups, or unrestricted administrative access.

Credential revocation must be practical. If an aircraft is lost, stolen, or suspected of compromise, the organization should be able to disable its remote access and invalidate relevant credentials without disrupting unrelated devices more than necessary. A recovery plan should also account for replacement hardware and legitimate maintenance, so that security controls do not encourage operators to share permanent credentials as a workaround.

Secure session management matters as well. Remote services should limit session duration where appropriate, invalidate sessions after relevant credential changes, and prevent unauthorized reuse of expired access tokens. Long-lived credentials may simplify unattended operation, but they increase the consequences of exposure. The design should balance operational requirements with the ability to restrict access when conditions change.

Encryption and authentication should be accompanied by integrity checks and replay protections appropriate to the protocol. A valid request may become unsafe if it is delayed and accepted after the aircraft's state has changed. Remote command systems should therefore consider message freshness, unique transaction identifiers, sequence information, or other supported controls. The exact mechanism should be consistent with the protocol and

implementation rather than added informally in ways that introduce compatibility or security defects.

A further concern is end-to-end protection. Data may be encrypted between the aircraft and a cloud service but then stored or processed by that service. The encryption protects the transmission path, not necessarily the data after the service has decrypted it. Organizations should understand where data is decrypted, which services process it, and which administrators or integrations can access it. Sensitive information should be protected throughout its lifecycle, including storage, analysis, export, and deletion.

## Cloud Relays, Remote Management Platforms, and Application Programming Interfaces

Cloud services can simplify the operation of fleets by providing a common place to collect telemetry, manage aircraft records, distribute approved mission data, and support remote monitoring. A cloud relay may receive information from an aircraft and forward it to one or more authorized clients. A remote management platform may also provide user accounts, device registration, software inventory, alerts, and operational reporting.

These services introduce additional trust relationships. The aircraft may trust a cloud endpoint to deliver messages, the cloud platform may trust registered devices to submit telemetry, and operators may trust the platform to display accurate information. Each relationship should be established through explicit authentication and authorization rather than inferred from network reachability or the possession of an ordinary account.

A cloud relay should have a clearly defined role. If it only forwards telemetry, it should not require unrestricted permission to modify flight parameters. If it supports command delivery, it should authenticate the requesting user, verify the target aircraft, enforce the relevant permissions, and record significant actions. The aircraft should still validate incoming commands independently rather than assuming that every message forwarded by the cloud platform is safe.

Device registration should establish a reliable relationship between the physical or logical aircraft identity and its authorized service credentials. The registration process should prevent unauthorized parties from claiming an existing device identity or registering equipment without approval. It should also support removal of retired or compromised devices, so that old credentials do not remain valid indefinitely.

Remote management platforms frequently expose application programming interfaces that allow software to retrieve telemetry, update records, manage devices, or issue operational requests. These interfaces can support automation and integration with other systems, but they also create a security boundary that must be protected. An API should define which operations are available, which identities may invoke them, what data each identity may access, and how invalid requests are handled.

Authentication tokens should be scoped to the minimum required capabilities. A service that only reads telemetry should not possess a token that permits mission modification or command transmission. Separate credentials for different applications and functions make it easier to identify misuse and revoke a single integration without disabling the entire platform.

Authorization should be enforced on the server side. Hiding an administrative function from a user interface does not prevent an

unauthorized client from attempting to call the underlying API directly. The server should verify permissions for each protected operation and ensure that a user cannot access another operator's aircraft merely by changing an identifier in a request. Object-level authorization is especially important in fleet systems where many users share the same platform.

API inputs should be validated according to the expected data types, sizes, and permitted values. Requests containing unexpected fields, malformed data, invalid identifiers, or unsupported operations should be rejected predictably. Error responses should provide enough information for legitimate troubleshooting without disclosing credentials, internal configuration, or sensitive system details.

Rate limits and resource controls can help protect APIs against accidental overload and certain forms of abuse. The limits should reflect legitimate operating patterns so that routine telemetry does not trigger unnecessary failures. Where commands and telemetry share a service, the architecture should prevent a large volume of low-priority data from exhausting resources needed for important control-related operations.

Cloud platforms also create a dependency on external availability. A service outage may prevent operators from viewing fleet status or sending remote updates even when the aircraft's cellular connection remains healthy. The system should identify which functions require cloud access and which can continue locally. Where operationally appropriate, the aircraft should retain approved mission information and maintain its configured safety behavior without assuming that a remote service will always be available.

Data governance is another important responsibility. A cloud platform may retain telemetry, flight logs, operator identities, mission files, and diagnostic records. The organization should understand the storage location, access model, retention period,

backup policy, and deletion process for this information. It should also determine whether external integrations receive copies of the data and whether those integrations are still necessary.

Cloud audit logs should record important administrative and operational events. These may include device registration, changes to user permissions, creation of API credentials, mission uploads, remote command requests, and modifications to security settings. Logs should be protected against unauthorized changes and should avoid exposing access tokens or cryptographic secrets. When a security incident occurs, reliable records can help establish which account or service performed an action and which systems were affected.

The security of third-party integrations deserves careful consideration. A fleet platform may connect to mapping services, analytics systems, maintenance databases, or storage platforms. Each integration may receive some portion of the operational data. The organization should limit the information shared with each service, review the permissions granted, and revoke access when the integration is removed. A trusted primary platform can still be exposed through a poorly secured secondary integration.

Remote administration through a cloud platform should be separated from routine monitoring where practical. Reading a battery report and changing a flight-related parameter are different levels of authority. A well-designed service should make these distinctions explicit and apply stronger safeguards to operations with greater consequences. Sensitive changes may require additional confirmation, an approved maintenance state, or a separate authorization step, depending on the aircraft's design.

The platform's behavior during account compromise should also be considered. If an operator's account is taken over, the organization should be able to suspend access, invalidate active sessions, review recent actions, and determine whether mission files or device

configurations were changed. Account recovery procedures must verify identity adequately; otherwise, an attacker may regain access through the recovery process even after the original credential has been revoked.

## Evaluating Cellular Connectivity Risks Without Interfering With Live Networks

A cellular security assessment should examine the aircraft's modem, onboard networking, remote services, and command-processing architecture without disrupting public mobile networks or interfering with other users. Testing must remain within an explicitly authorized scope. Mobile infrastructure is shared, and actions that generate unusual radio traffic or attempt to manipulate network behavior can affect devices and services beyond the system being assessed.

The safest starting point is a controlled laboratory configuration using an aircraft simulator, development hardware, an approved modem, and a test endpoint. The test environment should reproduce the communication path sufficiently to evaluate application behavior while avoiding unauthorized interaction with carrier infrastructure. Where specialized cellular-network testing is necessary, it should use an appropriately authorized private test network or a dedicated laboratory arrangement managed by qualified personnel.

The assessment should begin with a documented architecture. This should identify the modem, subscriber identity mechanism, companion computer, flight controller, remote endpoint, cloud services, and the routes used to exchange data. The assessor should record which functions depend on cellular connectivity and which should remain available when the network is lost. This provides a basis for evaluating both security and operational resilience.

Configuration review can identify many weaknesses without active testing. The review should examine modem firmware, network profiles, exposed management interfaces, device permissions, routing rules, firewall policies, remote-access settings, and the software responsible for forwarding telemetry or commands. It should also identify any default credentials, unnecessary services, or unrestricted interfaces that could provide access beyond the modem's intended role.

Subscriber management should be assessed through authorized administrative procedures. The organization should verify that subscriptions are assigned to approved devices, that account access is restricted, and that a lost or retired modem can be disabled through the appropriate provider or management process. The review should also consider whether the organization can distinguish active subscriptions from obsolete ones and whether subscriber records are handled securely.

Connectivity testing should establish the behavior of the system under ordinary conditions before testing failure scenarios. The assessor can record connection establishment, telemetry delivery, command acknowledgement, latency, data loss, and recovery behavior using approved test endpoints. These measurements provide a baseline against which controlled changes can be compared.

Network interruption can then be simulated in the laboratory by disconnecting the modem, disabling the test interface, or otherwise removing connectivity through approved methods. The assessment should observe whether the ground control application reports the loss accurately, whether stale telemetry is identified, and whether the aircraft maintains its configured behavior. After connectivity returns, the system should not automatically execute outdated commands or assume that its previous session remains valid without the required checks.

Authentication and encryption should be tested using the system's documented configuration and supported tools. The assessor should confirm that the remote service validates client identity, that the client validates the intended server, and that unauthorized requests are rejected. Where the system uses cryptographic signing, the test should verify the configured acceptance policy for signed and unsigned messages. These tests should be performed against endpoints under the assessor's control, not against unrelated mobile or cloud infrastructure.

Remote command authorization can be evaluated using simulated or nonhazardous test operations. The purpose is to establish whether the system distinguishes between users and services that may monitor telemetry and those permitted to request changes. A test account with read-only access should not be able to modify a mission or invoke a protected operation. Similarly, a service token intended for telemetry retrieval should not provide broader capabilities than its assigned role.

API assessment should focus on documented endpoints and authorized accounts. The review should examine authentication, authorization, input validation, rate limits, session management, and audit logging. It should verify that one user's credentials cannot be used to access another user's aircraft records without permission and that sensitive operations are logged. Testing should avoid destructive actions or requests that could affect devices outside the controlled environment.

The assessment should also evaluate data exposure. The organization should determine which telemetry fields are transmitted, whether the information is encrypted in transit, which cloud services receive it, and how long it is retained. Test data should be used wherever possible, especially when evaluating logs, analytics, or remote storage. If real operational data must be used, access should be

restricted and handling should follow the applicable privacy and security requirements.

Availability testing should remain controlled and proportionate. The assessor can introduce representative delays, packet loss, or temporary connection interruptions in a laboratory environment to observe how the system responds. The objective is to identify whether essential communication becomes unreliable, whether the application consumes excessive resources while reconnecting, and whether recovery creates unsafe state changes. High-volume testing against live carrier systems is not necessary to establish these basic properties and should not be conducted without explicit authorization from the relevant parties.

The assessment should include a review of failover behavior where the aircraft uses more than one connection. A system may switch from one cellular operator to another, move between cellular and local radio communication, or fall back to a locally stored mission. The transition should preserve the intended security policy. A secondary connection must not bypass authentication, expose an otherwise isolated service, or allow commands from a less trusted source simply because the primary path is unavailable.

Operational records should support investigation without creating unnecessary privacy risks. The system should record meaningful events such as authentication failures, connection interruptions, modem restarts, changes to network configuration, rejected remote commands, and changes to access permissions. Logs should use consistent timestamps where practical and should be protected from unauthorized modification. They should not contain subscriber secrets, passwords, signing keys, or full access tokens.

Findings should be documented with enough detail to reproduce the observed behavior in the authorized environment. Each finding should identify the affected component, the configuration under review, the conditions required to observe the issue, the expected

behavior, and the actual result. The report should distinguish confirmed defects from theoretical concerns and should explain the operational impact without overstating what the evidence demonstrates.

Remediation should be followed by repeat testing. A revised firewall rule may restrict an unnecessary service but inadvertently interrupt telemetry. A new authentication requirement may improve security while exposing a weakness in credential recovery. A modem firmware update may correct one problem but change compatibility with the onboard software. Regression testing verifies that the correction addresses the original finding while preserving required functions.

The final assessment record should also identify limitations. A laboratory test cannot prove that every mobile network will provide identical performance, nor can a short evaluation establish that every cloud integration is secure. The report should state which components and scenarios were tested, which were excluded, and what additional validation may be necessary before deployment. This disciplined approach allows an organization to improve remote drone connectivity without creating unnecessary risks for live networks, unrelated users, or the aircraft's own operational systems.

# Satellite Links and Beyond-Visual-Line-of-Sight Operations

## Satellite Communication Architecture for Unmanned Aircraft

Satellite communication extends the operational reach of unmanned aircraft beyond the practical limits of conventional line-of-sight radio links. In a traditional remotely piloted aircraft system, the operator generally maintains a direct radio connection with the aircraft, subject to antenna orientation, terrain, radio propagation, interference, and regulatory restrictions. As the distance between the aircraft and its control station increases, maintaining a reliable direct connection becomes more difficult. Satellite communications address this limitation by providing a communication path between the aircraft and a remote ground station through space-based infrastructure. This capability supports operations over oceans, remote industrial facilities, extensive agricultural areas, isolated infrastructure, and other locations where terrestrial communication networks may be unavailable or unsuitable.

The term beyond visual line of sight (BVLOS) describes operations in which the aircraft is flown beyond the remote pilot's direct unaided visual observation, subject to the applicable regulatory definition. BVLOS is an operational concept rather than a particular communication technology. Some BVLOS aircraft use long-range terrestrial radio, cellular networks, satellite links, or combinations of

these technologies. Satellite communication can make extended-range operations possible, but it does not independently establish that an aircraft is safe to operate beyond visual line of sight. The complete system must also address navigation integrity, airspace awareness, detect-and-avoid capability, command authority, emergency procedures, and the reliability of the aircraft's autonomous functions.

A satellite-connected unmanned aircraft system is best understood as a distributed computing and communication environment. Its major elements may include the aircraft's flight-control system, an onboard satellite terminal, an antenna, a satellite constellation or spacecraft, one or more ground gateways, terrestrial network infrastructure, a mission-control application, and the operator's control station. Depending on the service architecture, communications may pass through several network providers and data-processing facilities before reaching the intended destination. Each component introduces dependencies that affect performance, security, and operational continuity.

The onboard satellite terminal converts information from the aircraft's communication interfaces into signals suitable for transmission through the supported satellite network. Depending on the design, the terminal may carry mission telemetry, health information, command traffic, payload data, or selected operational messages. The flight controller may communicate with the terminal through a companion computer, a dedicated communication processor, or an integrated system interface. The precise arrangement varies by aircraft and terminal design, but the underlying principle remains the same: the aircraft's internal systems must exchange information with the external communication service through a controlled boundary.

This boundary deserves particular attention. A satellite terminal should not automatically receive unrestricted access to flight-control functions merely because it is a legitimate component of the aircraft.

Its permissions should correspond to its operational role. A terminal that transmits aircraft position and system health may require considerably less authority than a component responsible for relaying authenticated flight commands. Separating telemetry functions from command authority reduces the consequences of a compromised interface or a malfunctioning communication module.

The satellite link itself is only one part of the communication path. Information transmitted from the aircraft may travel through a satellite to a ground gateway, enter a terrestrial network, pass through a routing or security service, and reach a remote mission application. The return path may follow a similar route, although network architectures can use different gateways or routing arrangements in each direction. A communication system must therefore account for the security and reliability of the complete path, rather than treating the satellite transmission as the sole point of concern.

Satellite networks differ in orbital altitude, coverage, capacity, architecture, and expected communication delay. Geostationary satellites operate at an altitude of approximately 35,786 kilometres above the equator and appear nearly stationary relative to a fixed point on Earth. Their wide coverage can be useful for operations over large geographic areas, but the long signal path introduces substantial propagation delay. Low Earth orbit satellites operate much closer to Earth and can offer lower propagation delay, although their movement relative to the aircraft and ground infrastructure introduces different handover, coverage, and network-management requirements. Medium Earth orbit systems occupy an intermediate range and have their own performance and coverage characteristics.

These orbital categories should not be treated as guarantees of a particular service quality. Actual performance depends on the network design, terminal capabilities, congestion, gateway placement, routing arrangements, antenna performance, weather-

related effects where applicable, and the service agreement. A low Earth orbit connection may provide relatively low latency while still experiencing interruptions during coverage transitions. A geostationary connection may offer broad and stable coverage while being unsuitable for functions that demand extremely rapid feedback. Engineers must evaluate the specific service under realistic operating conditions rather than relying solely on an orbital label.

Satellite communications also differ in the way they allocate bandwidth and manage traffic. Some services are designed primarily for broadband data, while others prioritize low-rate telemetry, tracking, or safety-related messages. A system intended to transmit high-resolution inspection imagery may have very different requirements from one that needs to communicate aircraft position, battery condition, and a small number of control messages. Treating all data as equally important can create avoidable congestion. A more resilient architecture identifies which information must be delivered promptly, which can tolerate delay, and which can be stored for later transmission.

Command traffic and telemetry require especially careful treatment because their operational meaning depends on freshness and context. A message that accurately reports the aircraft's position several seconds ago may no longer be suitable for a decision that requires current position data. Similarly, a valid command may be unsafe if it arrives after the aircraft has entered a different flight mode or passed a planned decision point. The communication design must therefore consider timestamps, sequence information, message validity periods, and the current state of the aircraft, not merely whether a packet has reached its destination.

Satellite communication architecture should also distinguish between transport reliability and application correctness. A network may successfully deliver a message without proving that the message was generated by an authorized operator, remains appropriate for the aircraft's present state, or represents the intended action.

Network acknowledgements confirm particular aspects of delivery; they do not establish that a command is safe to execute. Authentication, authorization, application-level validation, and flight-envelope checks remain necessary even when the underlying satellite service is functioning normally.

The onboard system should preserve this distinction through clearly defined interfaces. Flight-critical functions should remain governed by the flight controller's safety logic, while the satellite terminal and remote applications provide communication and mission-level inputs within assigned limits. This arrangement helps prevent a failure in an external communication service from directly overriding the aircraft's fundamental control protections. It also makes it easier to test each component independently and identify which subsystem is responsible when an operational fault occurs.

## Ground Terminals, Gateways, and Remote Connectivity Dependencies

Ground terminals and gateways connect satellite communication services to the terrestrial systems used by operators, fleet managers, and mission applications. Although the aircraft terminal receives much of the attention in an airborne system, the ground segment can be equally important to security and availability. A well-protected aircraft can still be exposed to operational disruption if the ground terminal is misconfigured, the gateway is unavailable, the remote management service is compromised, or the operator's account has excessive privileges.

A ground terminal may include an antenna system, radio-frequency equipment, a modem, network interfaces, and software responsible for establishing and maintaining the satellite connection. In fixed installations, the equipment may be located at a permanent control

centre or operational facility. Mobile terminals may be installed on vehicles, vessels, temporary command stations, or portable ground-control equipment. Each arrangement introduces different physical and environmental concerns. Fixed installations can benefit from controlled access and stable power, while mobile installations may face greater exposure to theft, vibration, environmental damage, unstable power supplies, and untrusted local networks.

The terminal should be treated as a networked computing device rather than as a passive antenna. Its management interfaces, firmware, administrative credentials, configuration files, diagnostic services, and update mechanisms may affect the security of the wider communication system. Where management functions are exposed through an IP network, the terminal should be protected by access controls and network restrictions appropriate to its role. Administrative access should not be available to every device sharing the same local network, and unnecessary services should be disabled when doing so does not interfere with required operation.

A ground terminal's physical security is also relevant. Someone with unauthorized access to a terminal may be able to disconnect equipment, alter configuration, attach unapproved devices, or attempt to access administrative interfaces. The consequences depend on the terminal's design and the permissions it has within the wider network. Physical access does not automatically provide control of an aircraft, but it can undermine safeguards if sensitive interfaces or credentials are insufficiently protected. Secure enclosures, controlled equipment rooms, tamper-aware procedures, and documented maintenance practices reduce these risks.

Ground gateways serve a different function. They provide the connection between satellite infrastructure and terrestrial communication networks. Depending on the service, a gateway may handle routing, traffic management, network authentication, monitoring, or other functions required to carry data between the satellite segment and external systems. The gateway may be operated

by the satellite service provider rather than by the drone operator. This division of responsibility means that the aircraft operator may depend on security controls and availability commitments outside direct organizational control.

Such dependencies should be documented before an aircraft is approved for a particular mission. Operators need to understand which party manages the terminal, who is responsible for software updates, how outages are communicated, what service limitations apply, and which support arrangements exist for operational incidents. A service contract may describe availability objectives, but an objective is not the same as guaranteed uninterrupted connectivity. The operational plan must remain safe when a service becomes unavailable, even if the provider generally delivers reliable performance.

Remote connectivity can also involve cloud-hosted mission systems, fleet-management applications, authentication services, application programming interfaces, and data storage. These components may support mission planning, aircraft registration, telemetry visualization, remote configuration, maintenance records, or distribution of operational data. They improve coordination and allow operators to manage aircraft from distant locations, but they also introduce additional identities, permissions, software dependencies, and network boundaries.

A common architectural mistake is to assume that the satellite network itself provides sufficient protection for everything carried over it. Satellite transmission is a transport mechanism, not a complete security policy. Even where the provider encrypts a network segment or restricts access to its service, the operator may still need application-level encryption, message authentication, account controls, and authorization checks. Protection should be designed for the full communication path, including the aircraft, terminal, gateway, terrestrial network, cloud service, and operator interface.

Network segmentation is an important defensive measure. The equipment responsible for flight-critical communication should not automatically share unrestricted access with office computers, guest wireless networks, general-purpose file servers, or unrelated operational systems. Where practical, the architecture should separate management traffic, mission data, payload data, and safety-related communication according to their distinct requirements. Restrictions between these segments should be explicit, documented, and tested. Segmentation is not effective merely because different devices use different addresses; enforcement must prevent unauthorized traffic from crossing the intended boundaries.

The management plane deserves particular protection. It includes the functions used to configure equipment, create accounts, install updates, review logs, and change network settings. If an attacker gains management privileges, the attacker may be able to weaken controls without directly exploiting the satellite link. Administrative interfaces should therefore use strong authentication, restricted network access, and role-specific permissions. Where supported, sensitive administrative actions should be logged, and configuration changes should be reviewed or approved through a controlled process.

Credentials used by ground terminals and remote services should not be shared casually among operators or embedded permanently in configuration files. Shared credentials make it difficult to identify who performed a sensitive action and complicate revocation when personnel leave an organization or a device is lost. Individual accounts, managed service identities, protected credential storage, and defined credential-rotation procedures provide better accountability. Where machine-to-machine authentication is used, credentials should be scoped to the minimum privileges necessary and protected against extraction from software or configuration storage.

Software maintenance must cover the entire ground segment. Satellite terminals, gateways under the operator's control, mission applications, network devices, and remote management components may each have separate update processes. Operators should establish which systems receive security updates, how update authenticity is verified, how compatibility is tested, and how changes can be reversed if an update introduces an operational fault. Uncontrolled updates immediately before a flight can create risks, but indefinite postponement of security fixes can leave known vulnerabilities exposed. Maintenance should follow a documented process that balances operational stability with the need to address security weaknesses.

Ground connectivity also depends on ordinary infrastructure that may be overlooked during aircraft design. Power supplies, cooling, network cabling, backhaul connections, firewalls, identity services, and facility access controls can all affect service continuity. A gateway may remain operational while the operator's terrestrial connection fails. A mission application may be reachable while its authentication service is unavailable. A terminal may establish a satellite link while an incorrectly configured firewall blocks the traffic needed by the aircraft. Diagnosing these failures requires visibility into the complete service path, with monitoring that can distinguish a satellite-link problem from an application, identity, or local-network problem.

For security assessment, the ground segment should be examined as an interconnected set of trust relationships. The objective is to determine which components can communicate, which identities they trust, which actions each identity is authorized to perform, and how failures are detected. Authorized testing should use controlled equipment, documented permission, and non-disruptive methods. Production gateways and provider-managed infrastructure must not be probed or modified without explicit authorization. Where direct testing is unavailable, architecture reviews, configuration

inspections, service documentation, and simulated environments can provide meaningful evidence without placing live operations at risk.

## Latency, Availability, and Communication Failure Management

Latency is a defining consideration in satellite-connected aircraft operations. It describes the time required for information to travel through a communication system, while round-trip latency measures the time for a message to travel to a destination and for a response to return. The actual delay experienced by an operator can include propagation through space, processing within the satellite network, terrestrial routing, application processing, encryption, queueing, and display updates. The sum of these delays determines how current the operator's view is and how quickly a remote interaction can produce a visible response.

Propagation delay is especially significant for satellite systems with long signal paths. Signals must travel from the aircraft to the satellite and onward through the network, and a return command follows its own path. The physical distance imposes a minimum delay that software optimization cannot eliminate. Additional network and application processing increase the total. Engineers must therefore assess end-to-end performance instead of assuming that a fast onboard processor or a high nominal data rate will make remote control sufficiently responsive.

Not every aircraft function requires the same latency. A high-rate inner flight-control loop usually runs onboard, where sensor readings and control outputs can be processed locally with predictable timing. These loops stabilize the aircraft, adjust motor outputs, and respond to rapid changes in motion. Sending every stabilization decision through a distant satellite would introduce

delay and variability that could undermine control. Satellite communication is generally better suited to mission-level commands, telemetry, route updates, payload information, and supervisory functions whose timing requirements are compatible with the available service.

This distinction has direct safety implications. The aircraft should not depend on a continuous satellite connection to maintain basic stability or perform the immediate actions required to remain within its safe operating envelope. Its onboard control system should continue executing necessary control loops when remote communication becomes delayed or unavailable. Depending on the aircraft and mission, onboard logic may also need to maintain a safe route, monitor remaining energy, observe geofencing restrictions, and decide when a lost-link procedure must begin.

Latency is not simply a fixed number. It can vary with network congestion, satellite movement, handover events, gateway routing, atmospheric conditions affecting the particular frequency and link design, antenna obstruction, and equipment limitations. A system that performs well during a short demonstration may behave differently under operational loading or at a different geographic location. Mission planning should account for typical performance, expected variation, and credible worst-case conditions rather than using a single best-case measurement.

Jitter refers to variation in delay between successive messages. It can be as important as average latency because a system may receive information quickly for several seconds and then experience a substantial delay. For telemetry, jitter can make aircraft displays update unevenly. For command transport, it can produce unpredictable timing. Applications should distinguish recent information from stale information and should not interpret the arrival of an old message as evidence that the connection is currently healthy.

Message age is a useful operational concept. A telemetry message may be valid as a historical record but unsuitable as a representation of the aircraft's current state. Each important message should carry sufficient timing or sequence information for the receiving system to determine whether it is current, duplicated, delayed, or out of order. Where clocks are used, the design must account for synchronization limitations. Sequence numbers and explicit validity windows can help prevent old commands from being treated as fresh instructions, provided they are implemented correctly and protected against manipulation.

Availability describes the extent to which a service is usable when required. A satellite service may have strong average availability but still experience interruptions at critical moments. For an aircraft, the significance of an interruption depends on the mission phase, remaining energy, weather, airspace, distance from a safe landing location, and the functions affected. A short loss of a nonessential payload stream may have little safety impact. The same duration of lost command authority during a complex operation could require an immediate change in aircraft behaviour.

Availability should therefore be evaluated at the system level. The satellite service, onboard terminal, antenna, power supply, ground gateway, terrestrial backhaul, authentication service, mission application, and operator workstation can each become a single point of failure. Reliability calculations should reflect the actual architecture rather than assuming that the satellite link is the only relevant component. Redundant equipment can improve resilience, but only if the redundant paths do not share a common failure source that defeats both at once.

For example, two communication terminals connected to the same vulnerable power supply may both fail during a power event. Two network paths that rely on the same gateway or identity provider may not provide independent redundancy. A primary satellite

connection and a cellular backup can improve resilience in some environments, but the backup may be unavailable in remote regions or may have different latency, coverage, and capacity limitations. The engineering objective is not to maximize the number of connections, but to understand the independence, limitations, and transition behaviour of each path.

The aircraft must have a defined response to communication degradation and complete loss. A lost-link procedure specifies what the aircraft should do when it can no longer maintain the communication required for safe operation. Depending on the aircraft, approved operating concept, and surrounding conditions, the response may involve holding a safe state, continuing a pre-authorized route, returning to a designated area, diverting, or landing at a planned location. No single action is universally correct. Returning home, for example, may be unsuitable if the direct route crosses restricted airspace, the return point is no longer safe, or the aircraft lacks sufficient energy.

The procedure should define the conditions that trigger a response. These may include a period without authenticated commands, repeated failure to receive essential messages, excessive message age, degraded link quality, or loss of a required communication service. Trigger logic should avoid both extremes: reacting so aggressively to a brief harmless interruption that normal operations become unstable, and waiting so long that the aircraft loses the opportunity to perform a safe contingency. Thresholds should be validated for the actual aircraft and communication service.

Recovery behaviour is as important as failure detection. A connection may return after the aircraft has already entered a contingency mode. The system should not automatically resume unrestricted remote command execution simply because packets begin arriving again. It should establish the current aircraft state, confirm the identity and authority of the remote controller, reject expired or obsolete instructions, and apply the defined procedure for

restoring command authority. This avoids conflicting decisions between onboard automation and a remote operator who may not know which contingency action is underway.

Operators also need clear indications of degraded communication. A control station should distinguish between a healthy link, a slow or unstable connection, stale telemetry, loss of payload data, and loss of command capability. A generic green connection indicator may provide false reassurance if the underlying service is connected but the application is no longer receiving timely aircraft information. Useful indicators should reflect the functions that matter to safe operation and should communicate uncertainty without overwhelming the operator with irrelevant technical detail.

Monitoring should retain enough information to reconstruct the sequence of events surrounding an interruption. Relevant records may include link quality, message age, connection transitions, authentication outcomes, aircraft mode changes, contingency triggers, operator actions, and terminal or application faults. Logs must be protected against unauthorized modification and should use reliable time references where possible. Because telemetry and location records can reveal sensitive operational information, retention and access should also follow an appropriate data-protection policy.

Failure exercises should be performed in a controlled environment before operational deployment. Engineers can simulate delayed messages, packet loss, interrupted connectivity, invalid commands, gateway unavailability, terminal restart, and restoration of a communication path. These tests should confirm that the aircraft remains stable, the ground station reports the correct condition, and the recovery process does not create contradictory commands. Testing should include boundary conditions, such as an interruption that occurs during a change of flight mode or near a planned route decision point.

Where multiple communication paths are available, switching logic requires careful design. A backup path may have a different address, authentication method, bandwidth, or latency profile. The aircraft and ground station must agree on which path is active and how command authority is maintained during a transition. Duplicate commands, conflicting control sessions, and delayed messages from the previous path can create hazards if the system accepts them without validation. Failover should be treated as a safety-relevant state transition rather than a simple network reconnection.

## Identity Verification, Access Control, and Data Protection

Satellite communication does not remove the need to establish who is communicating with the aircraft. In a remote operation, messages may pass through multiple networks and service providers before reaching the destination. The system must distinguish legitimate aircraft components, authorized operators, approved mission applications, maintenance personnel, and external services from unknown or unauthorized entities. Identity verification and access control provide the basis for making these distinctions.

Authentication establishes that a device, service, or person is the identity it claims to be. Authorization determines what that authenticated identity is permitted to do. These are separate functions. An operator may successfully sign in to a mission application but still lack permission to change a flight plan or assume command of a particular aircraft. Similarly, a valid satellite terminal should not automatically be trusted to issue every type of command merely because it has established a connection to the network.

Aircraft and ground systems should use authentication mechanisms appropriate to their roles and capabilities. Cryptographic certificates, managed device identities, protected keys, and mutually authenticated connections can help establish trust between components. The choice depends on the system's technical constraints and operational requirements. Whatever method is selected, credentials must be generated, provisioned, stored, renewed, and revoked through a controlled lifecycle. A strong authentication design can be undermined if private keys are stored in easily extracted files or if default credentials remain enabled after installation.

Human access should be governed by individual accounts and role-based permissions. A pilot responsible for a particular mission may need authority to approve or execute flight commands, while a maintenance technician may need access to diagnostic information without permission to control an aircraft in flight. Fleet administrators may manage aircraft registration and configuration but should not necessarily have unrestricted operational command privileges. Dividing duties limits the consequences of an account compromise and helps prevent accidental changes by personnel whose responsibilities do not require them.

Privileged operations should receive additional protection. Actions such as transferring control, modifying operational limits, changing communication configuration, approving a mission, or installing software may warrant stronger authentication or explicit confirmation. For high-consequence actions, organizations may require approval by a second authorized person. Such controls must be designed so that they remain usable under real operating conditions and do not create unsafe delays during a genuine emergency. Emergency authority should be clearly defined and audited rather than handled through informal sharing of administrator credentials.

Multi-factor authentication can reduce the risk of account compromise, especially for remote access to mission-management systems. However, the mechanism must suit the operating environment. A remote operator may experience intermittent connectivity or may need to respond to a time-critical event. Authentication systems should be tested under these conditions so that a loss of access does not unexpectedly remove necessary capabilities. Resilience does not justify disabling security checks; it requires a carefully planned method for maintaining authorized access when an identity service is temporarily unavailable.

Command messages require protection beyond login authentication. A remote session may be established legitimately, yet individual messages can still be delayed, duplicated, altered, or replayed if the application protocol does not protect them adequately. Cryptographic integrity checks can help detect unauthorized changes, while sequence values, timestamps, session identifiers, and validity windows can reduce the risk of accepting obsolete messages. The exact implementation must be appropriate to the protocol and should be tested for synchronization errors, restart behaviour, and session recovery.

Encryption protects the confidentiality of information against unauthorized observation, but it does not by itself guarantee that a message is trustworthy or safe. An encrypted message may originate from a compromised authorized account. A validly authenticated command may still request an action that violates the aircraft's safety limits. Effective protection combines confidentiality, integrity, authentication, authorization, and application-level validation. These controls serve different purposes and should not be treated as interchangeable.

Key management is a particularly important issue in distributed systems. Cryptographic keys may be used by aircraft terminals, ground stations, mission services, and device-management

platforms. If one key is shared across many devices, a single compromise may affect the entire fleet. Where feasible, keys should be scoped to individual devices or clearly defined groups, and their use should be limited to the intended purpose. Compromised credentials must be revocable, and the system should have a documented method for restoring trust without requiring unsafe improvisation during an active mission.

Secure boot and verified software updates can strengthen confidence in the onboard and ground components that participate in remote communication. Secure boot checks whether software is authorized to execute during startup, while update verification checks whether a software package is authentic and has not been improperly modified. These mechanisms are most effective when supported by protected signing keys, documented update procedures, and controls that prevent unauthorized rollback to vulnerable versions. They do not eliminate every possible compromise, but they make certain forms of unauthorized software replacement more difficult.

Data protection extends beyond the commands used to operate the aircraft. Satellite-connected systems may transmit location histories, route plans, imagery, inspection findings, payload measurements, operator identities, maintenance records, and details about critical infrastructure. These records may have commercial, personal, security, or regulatory significance. An organization should identify which information it collects, why it needs the information, where it is processed, who can access it, and how long it should be retained.

Data minimization reduces unnecessary exposure. A mission application may not need to transmit raw payload imagery continuously if a smaller set of measurements is sufficient for operational monitoring. Diagnostic information may be stored locally and uploaded when bandwidth becomes available, provided doing so does not interfere with safety or required reporting. Reducing the volume of transmitted information can improve

bandwidth efficiency as well as limit the amount of sensitive data exposed if a service or account is compromised.

Encryption should be considered for data in transit and, where appropriate, data at rest. The satellite provider may protect part of the communication path, but the operator should understand precisely which segments are covered and which remain outside that protection. Data stored in a ground station, cloud service, maintenance laptop, or backup system may require separate safeguards. Encryption at rest is not a substitute for access control, because an authorized application or compromised account may still retrieve decrypted information.

Data location and third-party processing also require consideration. Some remote mission services may process information in facilities operated by external providers or in jurisdictions different from the aircraft's operating location. Organizations handling sensitive or regulated information should understand the relevant contractual terms, access arrangements, retention rules, and legal obligations. The goal is to make informed decisions about where operational data travels and which entities are permitted to handle it.

Audit records provide accountability, but they must be designed with care. Logs should capture important security events such as authentication failures, permission changes, control transfers, configuration updates, and unusual access patterns. They should also support reconstruction of safety-relevant events, including command acceptance and changes in communication state. Sensitive credentials and unnecessary personal information should not be written into logs. Access to records should be restricted, retention periods should be defined, and the integrity of important records should be protected.

A mature security design also plans for compromised identities. If an operator's credentials are suspected of theft, the organization should be able to suspend access, revoke relevant tokens or certificates,

review recent actions, and establish a trusted replacement identity. If an aircraft terminal is lost or removed from service, its credentials should not remain indefinitely valid. Revocation procedures must account for intermittent connectivity and the possibility that a disconnected aircraft cannot immediately receive updated trust information. These situations require predefined operational rules rather than assumptions that every device can contact a central service at all times.

## Security and Safety Requirements for Beyond-Visual-Line-of-Sight Operations

BVLOS operations place greater demands on system engineering because the remote pilot may not be able to observe the aircraft directly and may depend heavily on electronic information. Satellite connectivity can extend communication range, but safe operation depends on a broader set of capabilities. The aircraft must remain controllable during expected communication disruptions, the operator must receive sufficiently current information, and the operating organization must have procedures for managing hazards that cannot be resolved through ordinary remote commands.

Security and safety should be addressed together, while preserving the distinction between them. Safety engineering focuses on preventing unacceptable harm caused by system failures, environmental conditions, human error, or hazardous interactions. Cybersecurity focuses on protecting systems and information against unauthorized access, manipulation, disruption, and misuse. A cybersecurity event can create a safety hazard, such as when an attacker interferes with a command channel or falsifies telemetry. Conversely, a safety-related malfunction may create a security exposure by forcing the system into an unexpected state. A robust engineering process considers both directions of interaction.

The first requirement is a clear operational concept. Before selecting a satellite service or defining security controls, the operator should specify the intended mission, aircraft type, operating area, expected range, altitude, airspace conditions, environmental limitations, payload, control arrangements, and emergency options. These details determine the required communication performance and the consequences of a failure. A remote survey over an unpopulated area may have a different risk profile from a mission near roads, buildings, populated locations, or other aircraft.

The operational concept should identify the functions that depend on remote connectivity and those that must remain available onboard. Aircraft stabilization, immediate flight-control responses, and essential safety protections should not rely on a distant network round trip. Mission-level changes may be permitted remotely when the communication service meets the relevant performance requirements, but the aircraft should validate those changes against its current state and operating limits. A communication link should not be allowed to override onboard safety constraints simply because a command has been authenticated.

Risk assessment should examine credible hazards across the entire system. Relevant scenarios may include satellite service interruption, loss of a ground gateway, antenna obstruction, power failure, compromised operator credentials, corrupted navigation information, invalid mission updates, malicious or accidental command replay, application failure, and confusion during transfer of control. Each scenario should be assessed according to its likelihood, severity, detectability, and available mitigations. The assessment should consider combinations of failures where appropriate, because several individually manageable faults can become dangerous when they occur together.

A formal safety assessment should establish how the aircraft behaves when communication becomes degraded, when essential telemetry is

no longer current, and when the remote operator can no longer exercise the expected level of control. These behaviours should be defined before the mission begins and validated against the operational environment. The selected contingency may depend on the aircraft's energy reserve, route, weather, landing options, and airspace restrictions. The aircraft should not be expected to invent a safe response without adequate onboard logic and verified mission constraints.

Detect-and-avoid capability is another important consideration. Satellite connectivity does not give the remote pilot direct visual awareness of all surrounding traffic or obstacles. Depending on the aircraft and operating environment, BVLOS operations may require approved means of detecting and avoiding other aircraft, supported by appropriate surveillance information, onboard sensing, procedural separation, or a combination of methods. The suitability of each method depends on the operating concept and applicable rules. A satellite link can carry relevant information, but it cannot guarantee that the information is complete, current, or sufficient for collision avoidance.

Navigation integrity must also be considered independently of communication availability. An aircraft may maintain a strong satellite communication connection while experiencing unreliable positioning information. Navigation systems can be affected by interference, environmental conditions, equipment faults, or malicious manipulation. The flight-control system should monitor the health and consistency of navigation inputs and should have defined responses when the estimated position or velocity becomes unreliable. Position data received from an external service should not be treated as inherently trustworthy without appropriate validation.

Geofencing and route constraints can reduce the consequences of certain errors. A mission plan may define permitted operating boundaries, altitude limits, restricted areas, designated contingency locations, and routes to be used if a particular communication failure

occurs. The aircraft should verify mission data before execution and reject changes that violate the configured constraints. These protections should account for navigation uncertainty and should be tested for edge cases, including incorrect coordinate references, invalid altitude information, and inconsistent mission versions.

Regulatory compliance is a separate and essential requirement. BVLOS authorization depends on the jurisdiction, aircraft category, operational risk, airspace, and specific circumstances of the mission. Requirements may address operator qualifications, aircraft registration, operational authorization, command-and-control link performance, detect-and-avoid provisions, airspace coordination, emergency procedures, and recordkeeping. Satellite capability does not automatically grant permission to conduct BVLOS flights. Operators must identify the applicable aviation authority's requirements and ensure that the planned operation meets them before flight.

In Nigeria, operators should determine the requirements applicable to their proposed unmanned aircraft activity through the relevant Nigerian Civil Aviation Authority rules, directives, and authorization processes. Requirements can change, and the correct process may differ according to the aircraft, purpose, location, and operating profile. An organization should not assume that approval for one mission, aircraft, or operating area automatically authorizes a different BVLOS operation. Any required coordination with air traffic services, property owners, local authorities, or other stakeholders should be completed as applicable to the mission.

Communication performance requirements should be measurable. Instead of specifying that a satellite connection must be “reliable,” the engineering team should identify the functions that must remain available, acceptable message age, tolerable delay and jitter, expected coverage, interruption conditions, and the time available to enter a safe contingency. The values should come from the aircraft's design, hazard analysis, service characteristics, and operational

authorization rather than arbitrary universal thresholds. The requirements should be validated through representative testing and supported by evidence that reflects the actual configuration.

The safety case should address both normal operation and credible abnormal conditions. It should explain why the communication architecture is suitable, how failures are detected, what mitigations are provided, how the aircraft responds when connectivity is lost, and what evidence demonstrates that the chosen responses are adequate. Depending on the regulatory process, the evidence may include engineering analysis, simulation, ground tests, flight tests, maintenance records, and operational procedures. Each form of evidence has limitations. Simulation can explore many conditions efficiently, while controlled flight testing helps reveal behaviours that are difficult to represent fully in a model.

Cybersecurity verification should be integrated into this process. Authorized assessment can examine whether access controls are correctly enforced, whether communication messages are authenticated, whether expired commands are rejected, whether sensitive management interfaces are restricted, and whether software updates are verified. Testing should also evaluate whether the system reports failed authentication or abnormal traffic appropriately and whether a security event can trigger a safe operational response. The objective is not merely to demonstrate that an attack is difficult, but to establish that foreseeable security failures do not lead to unacceptable operational consequences.

Tests involving communication loss or command manipulation should begin in simulation, hardware-in-the-loop environments, or controlled ground facilities whenever feasible. Live flight testing introduces risks that must be justified and managed. Engineers should not deliberately disrupt public satellite services, interfere with licensed radio transmissions, or probe provider-managed infrastructure without explicit authorization. Tests should use approved equipment, suitable containment, documented procedures,

and clearly defined abort conditions. Where the relevant failure cannot be safely reproduced in flight, a combination of simulation, component testing, and architectural evidence may provide a more appropriate validation method.

Human factors must be incorporated into BVLOS planning. Remote operators may have to supervise aircraft over long periods, respond to delayed information, manage multiple alerts, and coordinate with other personnel while lacking direct visual contact. Control interfaces should present the aircraft's current mode, communication status, navigation confidence, and contingency state in a way that supports timely decisions. Critical alerts should be distinguishable from routine notifications, and the interface should avoid implying that the aircraft is healthy merely because one communication indicator remains active.

Procedures for transferring control require particular care. A transfer may occur between operators, control stations, communication networks, or organizations. The process should establish which party has command authority, confirm that the receiving party has the necessary information, and prevent simultaneous conflicting command sessions. A transfer should not be considered complete merely because a new operator has logged in. The system should have an explicit, verifiable state indicating whether control has been released, accepted, and established.

Operational procedures should also define who is responsible for responding to technical and security incidents. Personnel need to know how to report suspected credential compromise, unexplained changes in aircraft behaviour, persistent communication faults, invalid mission updates, or unexpected configuration changes. Response actions should protect the safety of an aircraft that may already be airborne. For example, revoking a credential may be necessary after a confirmed compromise, but the organization must also understand how that revocation affects an active mission and what safe control arrangements remain available.

Maintenance and configuration management are part of the safety requirements. The aircraft, satellite terminal, ground station, mission application, and supporting network should have identifiable approved configurations. Changes to firmware, network settings, authentication policy, or mission software can affect both security and operational behaviour. Significant changes should be reviewed, tested, documented, and approved before deployment according to their risk. Maintenance personnel should verify that the installed configuration matches the one evaluated for the operation and that temporary test settings have not been left active.

Finally, operational approval should rest on evidence that the complete system—not merely the satellite connection—can support the intended mission. The aircraft must retain safe local control, the communication architecture must meet defined requirements, identities and commands must be protected, the ground segment must be appropriately secured, and emergency procedures must remain workable under realistic conditions. These requirements need to be maintained throughout the system's service life, because satellite services, software, personnel, operating areas, and threat conditions can change after the initial assessment.

# Navigation Systems and Position Integrity

## Global Navigation Satellite Systems and Drone Position Estimation

Reliable navigation is fundamental to the safe and predictable operation of an unmanned aircraft. A drone must continually estimate where it is, how it is moving, and how its current state relates to the intended flight path. These estimates influence stabilization, route following, altitude management, geofencing, return-to-home behaviour, collision avoidance, and mission execution. When navigation information becomes inaccurate or unreliable, the aircraft may respond incorrectly even when its propulsion system, flight controller, and communication links remain fully operational. For this reason, navigation must be treated as a safety-critical function whose integrity deserves the same attention as flight-control reliability and communication security.

Modern unmanned aircraft commonly rely on Global Navigation Satellite Systems (GNSS) to estimate their position and velocity. GNSS is a general term covering satellite-based navigation services operated by different countries and organizations. These systems transmit timing and orbital information that compatible receivers use to calculate their position on Earth. A receiver estimates its distance from multiple satellites by measuring signal travel time and combining the resulting observations with information about satellite locations. Because the receiver's clock is not perfectly synchronized with the highly accurate clocks carried by navigation satellites, the calculation must also estimate receiver clock error.

Under suitable conditions, observations from at least four satellites allow the receiver to solve for three-dimensional position and clock bias.

The result is an estimated position rather than an infallible statement of the aircraft's true location. Its quality depends on satellite geometry, signal strength, atmospheric effects, receiver design, antenna performance, local obstructions, and the integrity of the information used in the calculation. The receiver may report a position under conditions in which the estimate is less accurate than usual. It may also produce plausible-looking data when the underlying observations are degraded. A navigation system therefore needs more than a coordinate output; it needs ways to assess the reliability of that output.

Several GNSS constellations can provide positioning services, and a receiver may support more than one. Multi-constellation reception can improve satellite visibility and help maintain navigation when buildings, terrain, vegetation, or the aircraft's orientation obstruct part of the sky. It can also improve the geometry of the available satellite observations. However, receiving signals from multiple constellations does not automatically guarantee integrity. Signals may be affected by common environmental conditions, receiver faults, interference, or errors in the processing chain. A system must evaluate the quality and consistency of its observations rather than assuming that a greater number of visible satellites always means a trustworthy position.

The distinction between accuracy and integrity is especially important in unmanned aviation. Accuracy describes how close an estimated position is to the true position. Integrity concerns the system's ability to provide trustworthy information and warn users when the information should not be relied upon. A navigation receiver may have good average accuracy while still failing to identify a rare but consequential error. Conversely, a receiver that detects degraded conditions and warns the flight-control system may

provide greater operational value than one that reports apparently precise coordinates without adequate fault detection.

Position integrity is therefore not adequately represented by the number of decimal places in a coordinate or by a single accuracy figure displayed on a ground-control screen. It involves confidence in the measurement process, the assumptions used by the navigation solution, and the ability to recognize conditions that make the result unsafe for its intended purpose. The level of integrity required depends on the operation. A drone collecting broad-area environmental imagery may tolerate more positional uncertainty than an aircraft approaching a narrow landing area or operating near an airspace boundary.

A GNSS receiver commonly provides additional information alongside position, including estimated velocity, satellite count, signal-quality indicators, timing information, and measures of expected positioning uncertainty. Some receivers also provide protection levels or integrity-related outputs, depending on their capabilities and supported service. These outputs must be interpreted according to the receiver's documentation. A generic estimate of horizontal accuracy should not be treated as a formal guarantee that the true position lies within the stated distance, particularly when the receiver does not provide the integrity monitoring required for that interpretation.

Position estimation also depends on the coordinate reference systems used by the aircraft and its mission software. Latitude, longitude, altitude, local north-east-down coordinates, and projected map coordinates represent different ways of describing position. Altitude is particularly easy to misunderstand because GNSS-derived height, height above mean sea level, and height above local ground are not interchangeable. A flight plan that combines incompatible altitude references can create operational errors even when the GNSS receiver is functioning correctly. Navigation interfaces should

make reference frames and units explicit, and the software should convert between them using validated methods.

The aircraft's position estimate must also be interpreted in relation to its attitude and movement. A multirotor can rotate or translate rapidly, and its antenna may experience changing orientation, partial obstruction, vibration, or electromagnetic interference from nearby equipment. An antenna installation that performs well on a stationary test bench may behave differently when mounted near power electronics, payload systems, or structural components. Antenna placement, grounding, cable routing, and electromagnetic compatibility should therefore be considered during the design and testing of the aircraft rather than left entirely to software configuration.

GNSS receivers also differ in their resistance to interference and their ability to detect inconsistent observations. More capable receivers may use multiple frequencies, advanced signal processing, or integrity-monitoring methods, but their performance still depends on the complete installation. A sophisticated receiver cannot compensate for every antenna defect, poor mounting arrangement, incorrect configuration, or faulty interface. The flight-control system should receive not only the position estimate but also relevant status information, timestamps, and validity indicators.

The way navigation data enters the autopilot is another important trust boundary. A receiver may connect through a serial interface, a dedicated navigation bus, or an intermediate computing module. The receiving software must check that messages are well formed, current, correctly identified, and within plausible limits. It should reject malformed data and detect unexpected changes in message timing or structure. Where supported, communication integrity checks can detect accidental corruption, but they do not necessarily establish that the underlying measurement is physically correct or that the transmitting device has not been compromised.

The navigation pipeline should consequently be understood as a chain of dependent operations: satellite signal reception, measurement processing, position and velocity calculation, quality assessment, data transmission, sensor fusion, and use by flight-control or mission-planning functions. A defect at any stage can affect the final estimate. Effective engineering documents this chain, identifies the assumptions made at each stage, and defines how faults are detected before they influence aircraft behaviour.

# Inertial Navigation and Multi-Sensor Data Fusion

GNSS provides an important external reference, but it is not sufficient for every navigation condition. Satellite signals may become unavailable indoors, near structures, under dense overhead cover, or in environments where interference degrades reception. Even in open areas, brief interruptions and measurement anomalies can occur. Inertial navigation provides a complementary source of motion information by estimating changes in the aircraft's movement from measurements made onboard.

An inertial measurement unit typically contains accelerometers and gyroscopes. Accelerometers measure specific force along their sensitive axes, while gyroscopes measure angular rate. These measurements help estimate how the aircraft is rotating and accelerating. After accounting for gravity, sensor orientation, and the chosen navigation frame, acceleration measurements can be integrated over time to estimate velocity and position. This process is valuable because it does not require a continuous external radio signal. Its principal limitation is that small measurement errors accumulate during integration.

An inertial sensor does not directly provide a perfect representation of the aircraft's position. Its measurements are affected by bias, scale-factor error, axis misalignment, temperature, vibration, noise, and mechanical stress. A small accelerometer bias can produce a growing velocity error, which then contributes to an even larger position error. Gyroscope errors affect attitude estimation, and attitude errors can cause acceleration to be projected incorrectly into the navigation frame. As a result, inertial navigation can perform well over short intervals while its uncertainty grows during extended periods without reliable external correction.

The rate of error growth depends on the quality of the sensors, the calibration process, the aircraft's motion, and the algorithms used to estimate the navigation state. Low-cost microelectromechanical sensors are attractive because they are compact, affordable, and energy-efficient, but their drift and noise characteristics must be understood. Higher-grade inertial systems can reduce some errors, although no sensor eliminates the need for calibration, monitoring, and fault management.

Sensor calibration establishes how the system's measurements relate to physical quantities. It may involve estimating bias, scale factors, axis alignment, temperature sensitivity, and other characteristics. Calibration quality affects the accuracy of both the inertial solution and the larger navigation system that uses it. Calibration values should be associated with the relevant hardware and configuration, and maintenance procedures should identify when recalibration or replacement is necessary. A calibration performed under one temperature or vibration condition may not fully characterize performance under every operating condition.

Inertial navigation becomes substantially more useful when combined with external measurements. Multi-sensor data fusion combines observations from sources with different strengths and failure modes to produce a more reliable estimate of the aircraft's

state. In a typical unmanned aircraft, the estimator may combine inertial measurements with GNSS position and velocity, barometric altitude, magnetometer measurements, optical-flow observations, range measurements, or vision-based navigation. The exact sensor set depends on the aircraft, its operating environment, cost constraints, and required performance.

A barometer estimates pressure and uses a pressure-to-altitude relationship to provide a useful vertical reference. Because atmospheric pressure changes with weather and location, barometric altitude can drift over time and should not be interpreted as an exact measurement of height above the ground. Propeller airflow, enclosure pressure, temperature, and rapid environmental changes can also affect the reading. A barometer can nevertheless complement GNSS and inertial measurements by providing a relatively smooth altitude estimate over short intervals.

Magnetometers measure the local magnetic field and can contribute to heading estimation. Their readings may be disturbed by electric currents, motors, steel structures, magnetic payloads, or nearby electronic equipment. A magnetometer that has not been calibrated or that is installed close to a strong magnetic source may produce an incorrect heading. Since heading influences the transformation of inertial measurements into a navigation frame, an incorrect magnetic reference can affect more than the displayed compass direction. The estimator should recognize when magnetic measurements disagree with other available information and should avoid treating them as universally trustworthy.

Optical-flow sensors estimate image movement relative to the aircraft and can support horizontal motion estimation when suitable visual texture and illumination are available. Their interpretation generally depends on altitude or distance information, because image motion alone does not establish absolute displacement in every situation. They may perform poorly over uniform surfaces, in low light, under motion blur, or when the observed scene moves

independently of the aircraft. Vision-based navigation can provide richer environmental information, but it too depends on scene characteristics, computational resources, and the quality of the camera data.

Range sensors can provide useful distance measurements to the ground or nearby surfaces. Their value depends on the technology used, the surface material, the angle of observation, environmental conditions, and the measurement range. A downward-facing range sensor may improve low-altitude control but cannot necessarily support navigation over terrain outside its valid operating envelope. The estimator should recognize when a measurement falls outside its specified range or conflicts with other observations.

Data fusion requires an estimator that represents the aircraft's state and accounts for uncertainty. Common approaches include Kalman filtering and related methods, complementary filters, and nonlinear state-estimation algorithms. These techniques differ in their assumptions, computational requirements, and handling of nonlinear relationships. Their purpose is not simply to average sensor values. They combine measurements according to their expected uncertainty and the relationships among the quantities being estimated.

A Kalman filter, for example, predicts the system's state using a motion model and then corrects that prediction using new measurements. The prediction step carries forward information from previous estimates and inertial observations. The correction step adjusts the estimate in response to external observations, weighted according to their expected uncertainty. Extended and unscented Kalman filters can accommodate certain nonlinear systems, while other estimation approaches may be appropriate for more complex models or different sensor arrangements.

Every estimator relies on assumptions about sensor noise, measurement timing, coordinate frames, and the behaviour of the

aircraft. If those assumptions are wrong, the resulting estimate may be unreliable even when the algorithm is implemented correctly. Incorrect noise parameters can cause a filter to trust a degraded sensor too strongly or reject valid measurements unnecessarily. Timing errors can lead to comparisons between observations that describe different moments. Coordinate-frame mistakes can produce systematic errors that resemble sensor faults.

The estimator should therefore monitor the consistency between its predictions and incoming measurements. A residual, sometimes called an innovation in filtering systems, represents the difference between a predicted measurement and the corresponding observation. Large or persistent residuals can indicate a sensor fault, unexpected motion, incorrect uncertainty assumptions, or an environmental disturbance. Residual monitoring must account for normal statistical variation and should not reject valid measurements solely because they differ from a prediction during an unusual but legitimate manoeuvre.

Sensor fusion also introduces the risk of common-mode failure. Two sensors are not necessarily independent simply because they are different devices. A GNSS receiver and a navigation application may both depend on the same underlying position source. Two receivers may share an antenna, power supply, data interface, or environmental exposure. A camera and optical-flow system may depend on the same image stream. If the shared element fails, both apparent sources of information may be affected simultaneously.

A robust architecture documents these dependencies and distinguishes genuinely independent evidence from duplicated outputs. Redundancy is most useful when the failure modes of the redundant components are sufficiently independent and when the system can identify disagreement. A second receiver connected to the same vulnerable power rail may improve availability for certain faults but provide little protection against a common power failure.

Similarly, two receivers exposed to the same local radio-frequency interference may fail together.

Sensor fusion should also preserve the difference between estimating a state and determining whether that state is safe to use. A filter may continue producing a numerical position after GNSS observations disappear, but the estimate's uncertainty may increase. The flight-control system should not treat the continued existence of an estimate as proof that it remains adequate for the current operation. Navigation validity should depend on the estimated uncertainty, sensor health, time since the last reliable external correction, and the requirements of the active flight mode.

## Understanding Navigation Errors, Sensor Drift, and Signal Anomalies

Navigation errors arise from many sources, and their operational consequences depend on how they affect the estimated state. Some errors are gradual and predictable, such as inertial drift. Others appear suddenly, such as a receiver reset, an antenna disconnection, or an abrupt inconsistency in position measurements. Still others may be intermittent or difficult to distinguish from legitimate aircraft motion. A reliable navigation system must account for these different patterns rather than relying on a single threshold for every fault.

GNSS errors can originate in satellite clock or orbit information, atmospheric delay, multipath, receiver noise, antenna performance, and local interference. Satellite navigation signals travel long distances and arrive at Earth with relatively low power, making them susceptible to environmental obstruction and certain forms of radio-frequency disturbance. Buildings, metal structures, and terrain can block signals or cause them to reflect before reaching the antenna.

Reflected signals may arrive later than direct signals and distort the receiver's timing measurements, producing position errors known as multipath effects.

Satellite geometry also influences the quality of the solution. Satellites clustered in a limited region of the sky provide less favourable geometry than satellites distributed across a wider range of directions. In poor geometry, measurement errors can have a larger effect on the calculated position. A receiver may still produce a valid-looking coordinate, so geometry indicators and uncertainty estimates should be considered when evaluating navigation quality.

Atmospheric conditions affect signal propagation. The ionosphere can delay signals in ways that vary with frequency and conditions, while the troposphere contributes additional delay. Multi-frequency receivers can estimate or reduce certain ionospheric errors, but atmospheric correction is not perfect in every circumstance. The magnitude of the effect depends on the signal frequencies, satellite geometry, receiver processing, and the environment. Engineers should use the performance specifications of the actual equipment rather than assuming that atmospheric effects are always negligible.

Interference can arise unintentionally from nearby electronics, poorly designed equipment, or other radio-frequency sources. Deliberate interference is also a recognized threat to satellite navigation. A receiver exposed to sufficient interference may lose satellite tracking, report reduced signal quality, or experience unstable navigation performance. More subtle conditions may not produce an obvious complete loss of signal. For defensive assessment, the important question is whether the system can detect degraded observations and transition to a safe operating mode without depending on a human operator noticing the problem first.

Spoofing is different from ordinary interference. Rather than merely making reception difficult, a spoofing source attempts to cause a receiver to accept misleading navigation signals. The resulting

position or timing information may appear plausible, which makes detection more challenging than detecting a complete loss of reception. This distinction is important for security engineering because availability monitoring alone does not establish position integrity. A receiver may continue reporting data even when the data are untrustworthy.

Navigation anomaly detection can examine consistency across independent sources, changes in signal characteristics, satellite measurement residuals, motion constraints, and agreement between GNSS and inertial predictions. No single indicator reliably detects every possible anomaly. An abrupt position jump may indicate a fault, but it could also result from an actual reset or an estimator transition. A gradual discrepancy may be caused by inertial drift, multipath, incorrect sensor calibration, or misleading external information. Effective monitoring uses several forms of evidence and considers the aircraft's operating state.

Inertial drift presents a different pattern. Gyroscope bias can lead to attitude errors, while accelerometer bias contributes to growing velocity and position uncertainty. Temperature changes may alter sensor behaviour, and vibration can introduce measurement noise or bias. If the estimator assumes that the sensors are more stable than they really are, it may become overconfident in a degraded inertial solution. Uncertainty modelling must therefore reflect measured sensor performance over the intended operating range.

Sensor saturation is another possible source of error. Accelerometers and gyroscopes have finite measurement ranges. If aircraft motion exceeds those limits, the reported values may no longer accurately represent the physical movement. Strong vibration, abrupt manoeuvres, or mechanical shock can also degrade measurements. The navigation system should monitor sensor status where available and recognize implausible or clipped data instead of continuing to process them as ordinary observations.

Timing errors can affect every stage of navigation. GNSS, inertial sensors, cameras, range sensors, and barometers may update at different rates. If observations are incorrectly timestamped or processed with inappropriate delays, the estimator may compare data from different physical moments. During rapid motion, even a modest timing offset can produce significant disagreement. Accurate time handling, documented sensor delays, and validated synchronization are therefore important parts of navigation integrity.

Communication faults can also corrupt the navigation information delivered to the flight controller. A message may be delayed, duplicated, truncated, or lost. An interface may report a previous valid position while the receiver is no longer updating. If the receiving software does not track message age, the aircraft may continue using stale information. Each navigation message should therefore be accompanied by sufficient timing and status information to determine whether it is current and valid for the active function.

The aircraft's physical behaviour provides another source of consistency checks. Estimated speed, acceleration, and attitude should remain broadly compatible with the available sensor measurements and known system limits. An abrupt coordinate change that implies an impossible acceleration may indicate a navigation fault. However, plausibility checks must be carefully designed because aggressive flight manoeuvres and emergency responses can produce unusual but valid motion. A detector that rejects every unexpected movement may be as unsafe as one that accepts every reported coordinate.

Navigation anomalies can also result from configuration errors. Incorrect coordinate reference frames, wrong sensor orientation parameters, stale calibration values, unit mismatches, or inappropriate estimator settings may create persistent errors without

any malicious activity. These failures can be especially difficult to diagnose because the system may behave consistently while remaining wrong. Configuration validation, commissioning tests, and independent checks of known positions can help reveal such defects before operational deployment.

A security assessment should not automatically classify every position disagreement as an attack. A defensible process distinguishes observed facts from suspected causes. Engineers should record the time of the anomaly, sensor status, navigation uncertainty, receiver diagnostics, aircraft mode, relevant environmental conditions, and subsequent system behaviour. The evidence may support a diagnosis of multipath, sensor drift, configuration error, interference, or suspected manipulation, but the conclusion should reflect the strength of the evidence. This discipline is essential for accurate incident reporting and for avoiding unnecessary changes to a functioning system.

## Assessing Position-Integrity Risks in Simulated Environments

Simulation provides a controlled means of examining navigation faults without deliberately exposing an aircraft to hazardous conditions. It allows engineers to reproduce sensor errors, communication delays, environmental constraints, and changes in aircraft motion while observing how the estimator and flight-control system respond. A well-designed simulation can reveal weaknesses that would be difficult, expensive, or unsafe to investigate through live flights alone.

A useful navigation simulation represents the relationship between the aircraft's true state and the measurements produced by its sensors. The true state may include position, velocity, attitude,

angular rate, and acceleration. Simulated sensors then generate observations based on models of their expected behaviour, including noise, bias, drift, update rate, latency, and measurement limitations. The navigation estimator processes these observations and produces its own estimate, which can be compared with the known simulated state.

This comparison allows engineers to measure estimation error over time. Horizontal and vertical position errors, velocity errors, attitude errors, and uncertainty estimates can be evaluated separately. The assessment should not focus exclusively on average error because a system may perform well under ordinary conditions while producing unacceptable errors during rare events. Peak error, duration of degraded performance, detection delay, and the time required to enter a safe state can be more important for certain missions.

The simulation should include representative operating conditions. Open-sky navigation, partial satellite obstruction, changing satellite geometry, urban multipath, rapid manoeuvres, temperature-related sensor variation, and different vibration levels may affect the system in different ways. Not every environment requires every scenario, but the test set should reflect the intended mission and credible hazards identified during risk assessment. Testing only under ideal conditions provides limited evidence of integrity.

Fault injection is one method of evaluating system behaviour. Engineers can introduce controlled changes to sensor data or system status to determine whether monitoring detects the fault and whether the aircraft responds appropriately. For example, a test may model a gradual increase in inertial bias, a GNSS outage, a delayed position message, a barometer offset, or disagreement between independent navigation sources. Each test should have a defined objective, expected detection behaviour, and acceptance criteria.

The goal is not simply to make the estimator fail. A useful assessment determines whether the system recognizes the change,

communicates the condition accurately, adjusts its uncertainty, and avoids using compromised information beyond its safe limits. A fault may be detected correctly but handled poorly if the system continues a flight mode that depends on the affected measurement. Conversely, a temporary sensor fault may be managed safely if the aircraft has sufficient independent information to maintain control while transitioning to an appropriate mode.

Simulation of GNSS anomalies should be conducted using approved simulation tools and isolated test systems. Engineers can model reduced satellite visibility, increased measurement noise, multipath-like errors, loss of satellite observations, and inconsistent position updates. More advanced test environments may model satellite geometry, receiver characteristics, atmospheric effects, and integrity-monitoring outputs. The model's assumptions and limitations should be documented so that simulated results are not mistaken for proof of performance under every real-world condition.

Security-oriented testing can evaluate whether the navigation system notices inconsistencies between external positioning and onboard motion estimates. For example, a simulation can introduce a gradual discrepancy between the simulated GNSS position and the position predicted from inertial measurements. The assessment can then measure the residual response, the detection threshold, the time to declare the GNSS data unreliable, and the subsequent flight-mode transition. This approach tests defensive behaviour without requiring the transmission of deceptive signals over the air.

A further test can examine recovery after a GNSS outage. The simulation can remove satellite observations for a defined period while allowing inertial sensors and other supporting measurements to continue. Engineers can observe how uncertainty grows, when the estimator declares the position unsuitable for a given function, and whether the aircraft changes its behaviour before the uncertainty exceeds the mission's acceptable limits. Once simulated GNSS data return, the test should assess whether the estimator accepts the

restored measurements appropriately and avoids an abrupt or unsafe state change.

The test environment should also examine common-mode failures. If two simulated GNSS receivers share a modelled interference condition, the system should not assume that their agreement proves correctness. If the primary and backup navigation sources rely on the same power supply or processing unit, the test should include the relevant shared failure. This reveals whether the architecture offers genuine independent evidence or merely duplicates a vulnerable source.

Hardware-in-the-loop testing can extend simulation by connecting real navigation or flight-control hardware to a simulated aircraft environment. The actual hardware processes simulated sensor data while the simulator models the aircraft's response. This arrangement can reveal implementation issues such as timing errors, processor load, interface failures, message parsing problems, and differences between an ideal software model and the deployed device. It is particularly useful when a safety response depends on the interaction among sensor drivers, estimator software, and flight-mode logic.

Software-in-the-loop testing provides a different level of evidence. It can run the navigation and control software within a simulated computing environment, making it easier to test many scenarios and automate repeatable experiments. It may not reproduce every hardware-specific behaviour, so it should be supplemented with other forms of testing when timing, electrical interfaces, sensor noise, or processor constraints are important. A layered validation strategy uses simulation efficiently while recognizing where physical testing is necessary.

Acceptance criteria should be established before test execution. Otherwise, engineers may unconsciously interpret results in a way that favours the system. Criteria may define acceptable position error, maximum detection time, permissible uncertainty growth,

correct rejection of invalid measurements, and successful entry into the required contingency mode. The specific values must be derived from the aircraft, mission, safety analysis, and applicable operational requirements. There is no single universal threshold appropriate for every unmanned aircraft.

Test results should be reproducible. Each test record should identify the software version, hardware configuration where applicable, sensor models, estimator parameters, scenario conditions, injected fault, expected outcome, observed outcome, and any deviations from the test plan. Reproducibility enables engineers to determine whether a software change has improved performance or introduced a regression. It also provides evidence for technical reviews and helps maintenance teams understand why a particular configuration was approved.

Simulation results must be interpreted carefully. A model may omit environmental effects, hardware limitations, receiver-specific behaviour, or unusual interactions that occur during actual flight. Passing a simulated test establishes performance under the tested assumptions; it does not establish that the system is immune to every navigation anomaly. Engineers should identify which risks are adequately covered by simulation, which require hardware validation, and which need controlled flight testing under an approved safety plan.

The most useful outcome of a simulation programme is not a collection of successful test runs but a clear understanding of the system's limits. Engineers should be able to identify the conditions under which the navigation estimate remains reliable, the conditions under which uncertainty grows, the point at which a function must no longer rely on the estimate, and the recovery behaviour expected when better information becomes available. These findings should inform the aircraft's configuration, operating procedures, maintenance requirements, and training.

# Designing Navigation Monitoring, Redundancy, and Recovery Mechanisms

Navigation monitoring converts raw sensor information and estimator outputs into decisions about whether the aircraft can safely rely on its current state estimate. A well-designed monitor considers the health of individual sensors, the consistency of their measurements, the uncertainty of the fused estimate, the freshness of the data, and the requirements of the active flight mode. It should provide clear status information to the flight-control system and operator without creating unnecessary mode changes in response to harmless fluctuations.

The monitor should distinguish between sensor availability and sensor validity. A receiver may be powered and communicating while its position estimate is unreliable. An inertial sensor may continue sending measurements while suffering from excessive bias or saturation. A magnetometer may produce stable readings that are inconsistent with the local magnetic environment. Availability indicates that a component is providing data; validity requires evidence that the data are suitable for the intended use.

A practical monitoring design combines several independent checks. These may include message freshness, valid sensor status, physically plausible measurement ranges, residual consistency, uncertainty limits, agreement among navigation sources, and operational constraints. Each check addresses a different failure type. Message-age monitoring can detect stale data but may not detect plausible false data. Residual checks can detect certain inconsistencies but depend on the quality of the estimator's motion model. Independent sensors can provide corroboration, but only if their failure modes are sufficiently separate.

Thresholds should reflect the system's operating context. A residual that is unusual during steady flight may be normal during a rapid manoeuvre. An uncertainty level that is acceptable for broad-area surveying may be unacceptable for a precision approach. The navigation monitor should therefore use thresholds justified by analysis and testing, and it should account for flight mode, aircraft dynamics, and mission requirements where appropriate. Thresholds should not be adjusted casually in response to individual incidents without understanding their wider consequences.

Uncertainty estimation is particularly important during periods without external position correction. An inertial estimator may continue producing a position estimate after GNSS loss, but its uncertainty should grow in accordance with the sensor characteristics and elapsed time. The aircraft should compare this uncertainty with the limits required for its current function. When the estimate no longer meets those requirements, the system should transition to a mode that does not depend on the compromised level of navigation accuracy.

This transition should be deliberate and predictable. A loss of GNSS does not necessarily require immediate motor shutdown or an abrupt landing. The appropriate response depends on whether the aircraft can maintain stable control, whether alternative navigation sources remain valid, and whether a safe route or landing option is available. The system may be able to continue temporarily using inertial and other measurements, but only within a defined uncertainty envelope. If the remaining navigation information is insufficient, the aircraft may need to hold where feasible, reduce mission complexity, proceed to a preplanned contingency location, or land in a suitable area.

Redundancy can improve navigation resilience, but it must be designed around independent failure detection and meaningful fallback capability. Two GNSS receivers can provide useful

comparison, especially when they have separate antennas, power arrangements, interfaces, or processing paths. Yet two receivers using the same satellite signals in the same environment may experience similar errors. Their agreement is useful evidence, not proof that the reported position is correct. The architecture should consider common exposure to interference, shared configuration errors, and common dependencies.

Combining GNSS with inertial navigation provides a stronger basis for continuity because the technologies have different characteristics. GNSS can constrain long-term position drift, while inertial sensors provide rapid motion measurements during brief satellite interruptions. The combination is not immune to failure: the inertial solution drifts, and GNSS can be inaccurate or misleading. The estimator must determine when the sources agree, when one should be down-weighted or rejected, and when the available evidence is insufficient to maintain the required navigation integrity.

Other sensors can provide additional information under suitable conditions. Barometric altitude, optical flow, visual odometry, range measurements, terrain-relative navigation, or other approved positioning methods may support particular aircraft and missions. Their usefulness depends on the environment and their own limitations. A camera-based method may perform well where visual features are stable and lighting is adequate, but poorly over uniform terrain or in darkness. A range sensor may support low-altitude control without providing reliable long-range position. The system should select backup sources based on validated capabilities rather than treating every available sensor as an interchangeable substitute for GNSS.

Redundancy also requires a clear selection strategy. If two navigation sources disagree, the system must determine whether one is faulty, whether both are degraded, or whether the difference can be explained by timing, coordinate-frame, or calibration issues. Automatically choosing the source with the smallest reported

uncertainty can be unsafe if that source is overconfident or has a fault that its internal diagnostics do not detect. Source selection should consider external consistency checks, known limitations, health indicators, and the consequences of choosing incorrectly.

The architecture should protect navigation decisions from unauthorized changes. Sensor configuration, estimator parameters, geofence definitions, mission routes, and navigation-source selection can affect flight behaviour. Access to these settings should be restricted to authorized personnel, and changes should be recorded and reviewed according to their risk. Mission data should be validated before use, including coordinate formats, reference frames, altitude conventions, and route boundaries. A corrupted or incorrectly formatted mission can cause unsafe behaviour even when every sensor is operating normally.

Software integrity matters as well. Navigation drivers, estimation algorithms, and flight-mode logic should be developed and maintained through controlled software processes. Updates should be authenticated, tested, and associated with identifiable versions. A change to a sensor driver may affect message timing or units, while an estimator change may alter how uncertainty is calculated or how measurements are rejected. Regression tests should therefore cover both ordinary navigation performance and known fault scenarios. Security reviews should consider whether unauthorized changes could disable monitoring or make the system accept invalid data.

Recovery mechanisms must account for the fact that navigation quality may improve gradually rather than returning immediately to normal. When GNSS reception resumes, the new measurements may disagree with the inertial estimate accumulated during the outage. The estimator should reconcile this difference in a controlled manner, considering the quality of the restored observations and the current uncertainty of the inertial solution. An abrupt position correction may produce a sudden change in the estimated state, so

the system should manage reinitialization and state correction in accordance with its validated design.

Recovery should not automatically cancel every precaution after a single valid measurement arrives. The system may require sustained evidence that the source is healthy, that observations are consistent, and that the navigation solution meets the applicable integrity requirements. Depending on the architecture, it may reintroduce the source gradually, reset certain estimator states, or require an explicit operator acknowledgement before resuming a higher-dependency flight mode. These behaviours should be specified and tested rather than left to undocumented implementation details.

The operator's interface should communicate navigation status in operationally meaningful terms. Instead of presenting only raw satellite counts or a generic position icon, it should indicate whether the current navigation solution is suitable for the active mode, whether a sensor is degraded, whether uncertainty is increasing, and whether a contingency procedure has been triggered. Technical details remain useful for maintenance and engineering, but flight crews need information that supports timely decisions without requiring them to interpret complex sensor diagnostics while managing an aircraft.

Alerts should distinguish warnings from conditions that require immediate action. Excessive alerting can cause operators to overlook a critical message, while vague warnings may not communicate the significance of a navigation problem. Alert design should consider the action required, the time available, and the operator's existing workload. Where the aircraft can manage a condition autonomously, the interface should show the action being taken and the conditions required for recovery. Where human intervention is necessary, the alert should state the relevant limitation and the available approved response.

The system should also preserve enough diagnostic information to support investigation after a navigation incident. Records may include GNSS quality indicators, satellite observation status where available, inertial measurements, estimator residuals, uncertainty estimates, sensor-health flags, mode transitions, and operator actions. These records can help distinguish a sensor defect from an environmental disturbance, configuration problem, software regression, or suspected security event. Logging should be designed to avoid excessive computational load and should protect sensitive operational data against unauthorized access or modification.

Maintenance procedures should include checks of navigation antennas, cabling, mounting, sensor alignment, vibration isolation, calibration, power quality, and relevant software configuration. A receiver replacement or antenna relocation may change performance even if the replacement component is nominally equivalent. After significant maintenance, the organization should perform appropriate verification to establish that the navigation system behaves as expected. Any change affecting the safety assumptions of the aircraft should be reviewed before the system returns to operational service.

Operational limits should reflect the verified capability of the navigation system. If a particular mission requires a level of position integrity that the aircraft cannot maintain under expected conditions, the correct response is to change the mission, improve the equipment, or establish a different approved operating method. Procedures should not depend on an operator compensating indefinitely for a known navigation weakness. Safe operation requires a defensible relationship between the system's measured capabilities, the risks of the intended flight, and the actions available when those capabilities deteriorate.

# Companion Computers and Payload Security

## Companion Computers and Their Role in Autonomous Flight

Modern unmanned aircraft increasingly rely on computing resources that extend beyond the primary flight controller. While the flight controller manages immediate stabilization, actuator control, and essential flight-state functions, a companion computer can provide the processing capacity required for higher-level autonomy, mission coordination, sensor interpretation, and communication with external systems. This separation allows an aircraft to perform sophisticated tasks without placing every computational demand on the processor responsible for maintaining stable flight. It also introduces additional software, interfaces, privileges, and dependencies that must be managed carefully.

A companion computer is an onboard computing platform that operates alongside the flight controller. Depending on the aircraft's design, it may be based on a single-board computer, an embedded processor module, an industrial computing unit, or another compact system suitable for airborne operation. Its capabilities may include running an embedded operating system, processing camera feeds, performing object detection, managing mission applications, collecting payload data, and communicating with remote services. Some platforms also coordinate multiple sensors or execute algorithms that support autonomous navigation and decision-making.

The distinction between a companion computer and a flight controller is primarily functional, although the precise boundary varies among aircraft architectures. A flight controller typically runs real-time firmware designed to respond predictably to sensor measurements and control demands. Its tasks may include attitude stabilization, rate control, motor output generation, state estimation, and the execution of safety-related flight modes. A companion computer usually operates with a broader software environment and greater processing flexibility. It may run complex applications, computer-vision libraries, databases, network services, and mission-specific code that would be difficult to accommodate within the flight controller's more constrained environment.

This division creates an important engineering advantage. Computationally demanding functions can operate on the companion computer while the flight controller retains responsibility for immediate control. A vision application, for example, might identify a landing marker or classify an observed surface. It can provide a proposed target or navigation-related observation to the flight controller, which then evaluates the input according to its own control logic and safety constraints. The application supplies information; it should not automatically gain unrestricted authority over the aircraft.

The separation becomes particularly valuable when applications are complex or frequently updated. Computer-vision software may require changes to machine-learning models, image-processing libraries, or operating-system packages. Mission applications may need new features or revised interfaces as operational requirements change. If these functions are separated from the primary flight-control software, they can often be maintained without modifying every component of the aircraft. Nevertheless, separation alone does not guarantee safety. The interface between the companion computer and flight controller must be designed so that failures or compromised applications cannot bypass the aircraft's essential protections.

A companion computer can support autonomous behaviour by combining data from several sources. It may process images, interpret mission instructions, compare observations against a planned route, communicate with a ground station, or coordinate a payload with aircraft position and attitude. More advanced systems may use onboard inference to identify objects, recognize environmental features, or evaluate whether a particular mission condition has been met. Such processing can reduce dependence on continuous communication with a remote operator, but it also increases the consequences of software errors and unreliable input data.

Autonomy should not be confused with unrestricted decision-making. A well-designed aircraft has a defined operational envelope that constrains what its software is permitted to do. A companion application may recommend a route adjustment, identify a potential landing area, or request a change in mission state, but the aircraft's control architecture should determine whether that request is valid. Requests should be checked against geofencing restrictions, altitude limits, energy reserves, navigation confidence, current flight mode, and other relevant safety conditions.

The trust boundary between the companion computer and flight controller is therefore one of the most important interfaces in the aircraft. If a companion application can issue commands that alter flight behaviour, the system must define which commands are permitted, when they may be accepted, and how the flight controller validates them. A compromised image-processing service should not be able to disable a failsafe or change protected control parameters merely because it shares a computer with a legitimate mission application.

This issue becomes more significant when several applications share the same companion platform. A camera service, telemetry collector, mission planner, remote-management agent, and diagnostic utility

may all operate simultaneously. If every application runs with broad system privileges, a weakness in any one of them could affect the others. The computer may remain physically separate from the flight controller while still becoming a single point of compromise for mission data, external communication, and command requests. Secure design must therefore consider both the hardware boundary and the permissions granted to individual software components.

The companion computer also creates a potential dependency on resources that were not central to traditional flight-control designs. Processor overload, memory exhaustion, storage failure, overheating, or a malfunctioning application may prevent the computer from providing timely information. If the aircraft depends on that information for a particular autonomous function, the loss must be handled explicitly. The flight controller should continue to maintain safe basic operation when the companion computer fails, provided the aircraft's architecture and approved operating concept permit such a response.

This requires clear allocation of responsibilities. Essential stabilization and immediate control should remain independent of nonessential companion applications wherever practical. If a companion computer provides a necessary input for a particular operation, the aircraft should define what happens when that input becomes unavailable or invalid. The response may involve suspending the affected function, continuing only within a reduced operating envelope, switching to a validated alternative, or entering a planned contingency mode. The appropriate response depends on the mission and aircraft design, but it should be established before flight.

Hardware reliability matters alongside software security. Airborne computers may experience vibration, temperature variation, restricted cooling, unstable supply voltage, electromagnetic interference, and mechanical stress. These conditions can cause intermittent faults that are difficult to reproduce on a development

desk. Secure and reliable companion-computer engineering includes appropriate power regulation, thermal management, storage selection, connector retention, grounding, and environmental testing. A computer that is vulnerable to routine electrical or thermal disturbances can become a source of operational risk even if its software is well protected.

The companion computer should also have a documented lifecycle. Its approved hardware configuration, operating-system image, application versions, model files, communication interfaces, and security settings should be identifiable. This makes it possible to reproduce tests, investigate incidents, and determine whether an unexpected behaviour resulted from a software change, hardware replacement, or configuration difference. Without reliable configuration records, troubleshooting can become an exercise in guesswork, particularly when several aircraft use slightly different onboard software.

## Embedded Linux, Application Isolation, and Onboard Services

Many companion computers use embedded Linux because it supports a broad range of processors, communication interfaces, development tools, and application frameworks. It can provide networking, file systems, process management, hardware access, and support for complex libraries within a relatively flexible environment. These capabilities are useful for autonomous aircraft, but they also introduce the responsibilities associated with operating a general-purpose computing platform. A companion computer may contain far more software than is necessary for its immediate mission, and each additional component can create maintenance demands or a potential security weakness.

An embedded Linux system commonly includes a bootloader, kernel, device drivers, system libraries, application services, configuration files, and user applications. The bootloader initiates the startup process and loads the operating system. The kernel manages memory, processes, device access, and communication between software and hardware. User-space services provide functions such as networking, time synchronization, logging, camera access, and application coordination. Each layer has its own security considerations, and a weakness in a lower layer can undermine protections implemented above it.

A secure design begins with a system image that contains only the components needed for the intended operation. Development utilities, unused network daemons, unnecessary remote shells, sample applications, and default demonstration services should not remain enabled without a clear operational justification. Reducing the software footprint makes the system easier to maintain and limits the number of services that must be protected. Minimalism does not mean removing useful monitoring or recovery capabilities; it means retaining necessary functions while avoiding unneeded exposure.

Accounts and privileges should be designed around specific responsibilities. Applications that process camera frames do not ordinarily need permission to change system-wide network settings. A telemetry collector may need access to selected status messages but not unrestricted access to stored credentials. A diagnostic service may require read-only access to certain system information without authority to alter flight configuration. Assigning separate identities and narrowly scoped permissions reduces the likelihood that a compromised application can affect unrelated functions.

Running every application as an administrator is particularly risky. Broad privileges make ordinary software defects more consequential because a flaw in one service may provide access to system files,

credentials, device interfaces, or other processes. Where feasible, applications should run under dedicated accounts with restricted file access, limited device permissions, and controlled network capabilities. Administrative access should be reserved for defined maintenance tasks and should not be required for routine application operation.

Application isolation provides another layer of protection. Processes normally have separate address spaces, but process separation alone does not prevent one application from accessing resources exposed through shared files, sockets, devices, or privileged services. Stronger isolation can be achieved through mechanisms such as restricted service identities, mandatory access controls, namespaces, capability restrictions, containers, or virtual machines where the hardware supports them and the added complexity is justified.

Containers can help package an application with its dependencies and restrict some aspects of its execution environment. They can improve deployment consistency and reduce accidental interaction between services. However, containers share the host kernel in common configurations, so a kernel vulnerability or an excessively privileged container can undermine the intended boundary. A container should not be treated as a complete security barrier merely because an application runs inside it. Its permissions, mounted directories, device access, network connectivity, and update process require careful review.

Virtual machines can provide a stronger separation in some designs by running software within distinct operating-system environments, but they impose additional resource and maintenance costs. On a compact aircraft, processing capacity, power consumption, boot time, and thermal limits may constrain the use of heavier isolation mechanisms. Engineers should select controls based on the sensitivity of the function, the consequences of compromise, and the capabilities of the platform. The strongest theoretical boundary is

not necessarily the best choice if it cannot be maintained or validated within the actual aircraft.

Onboard services require particular attention because they frequently connect otherwise separate components. A camera service may publish image data to a vision application. A telemetry service may distribute aircraft status to mission software. A network service may expose diagnostic information to a ground station. A local message broker may allow applications to exchange structured events. These services can simplify development, but they also become security-sensitive intermediaries.

Each service should have a clearly defined purpose, interface, and permission model. It should expose only the operations required by its legitimate clients. If a service only needs to deliver camera frames, it should not also provide unrestricted file access or system administration functions. If a diagnostic interface is needed only during maintenance, its availability should be restricted according to a documented procedure rather than left permanently accessible to any device on the network.

Network services should bind to the interfaces they actually require. A service intended for local communication between applications may not need to accept connections from the aircraft's external network interface. Where remote access is necessary, firewall rules and application authentication should limit which systems can reach it. A service that listens on every interface by default can become accessible through a wireless connection or remote network even when its original purpose was local processing.

Secure boot and verified software deployment can help establish trust in the computing environment. Secure boot mechanisms check whether startup components are authorized according to a configured trust policy. Signed software packages can help establish that an update came from an approved source and has not been modified. These protections depend on secure key management,

trustworthy update procedures, and resistance to unauthorized configuration changes. They do not automatically prove that approved software is free from vulnerabilities, but they can make unauthorized replacement more difficult.

Updates should be tested against the aircraft's operational requirements. An operating-system update may change device drivers, network behaviour, camera access, scheduling, or library compatibility. A new version of a computer-vision framework may alter processing time or memory consumption. An application that performs correctly on a development computer may exceed the companion computer's available resources during actual flight. Updates should therefore be validated under representative workload and environmental conditions before being deployed to operational aircraft.

A robust deployment process should maintain a known-good software configuration and a controlled recovery method. If an update fails or introduces an unexpected fault, technicians should be able to restore an approved state without bypassing essential security controls. Rollback mechanisms themselves need protection because unrestricted rollback can reintroduce vulnerable software. A secure design may allow recovery to a previously approved version while preventing installation of unauthorized or known-prohibited versions.

Logging and monitoring should provide enough information to identify unusual activity without exhausting limited onboard storage or processor resources. Relevant events may include failed authentication, unexpected service restarts, changes to critical configuration, abnormal resource consumption, and denied access to protected interfaces. Logs should be protected against unauthorized modification where feasible, and sensitive credentials should never be written into them. Retention policies should account for storage capacity, investigation needs, and the sensitivity of recorded operational data.

Resource exhaustion is an important concern on embedded platforms. An application that consumes excessive CPU time, memory, disk space, or network bandwidth can degrade the performance of other services. Even without malicious intent, a camera process may allocate too much memory, a logging service may fill the storage device, or an image-processing task may delay other applications. Resource limits, health checks, process supervision, and defined restart behaviour can reduce the consequences of these failures.

The operating system should also be configured to preserve predictable behaviour under load. Mission applications may need priority over nonessential background processing, but priority settings must be designed carefully. A poorly configured high-priority process can prevent other tasks from receiving enough processor time. Engineers should measure actual scheduling behaviour, memory pressure, and thermal performance rather than assuming that a nominally powerful processor will always meet real-time demands.

Timekeeping and synchronization are additional considerations. Camera frames, telemetry records, navigation messages, and payload measurements may need to be associated with a common time reference. Incorrect timestamps can lead to mismatched sensor observations, inaccurate event reconstruction, or misleading mission records. If the companion computer receives time from GNSS or a remote service, the system should handle loss of that source and distinguish synchronized time from an uncertain local clock. Security-sensitive operations should not depend blindly on an external time source without considering its trustworthiness.

# Cameras, Gimbals, Imaging Systems, and Payload Interfaces

Payload systems extend the purpose of an unmanned aircraft beyond flight itself. Cameras, thermal sensors, environmental instruments, mapping equipment, and other specialized devices collect information required by the mission. Their data may be processed onboard, transmitted to a ground station, stored for later analysis, or combined with navigation information. Although payloads may not directly control propulsion, their software and interfaces can introduce significant security risks because they often connect to the companion computer and may share power, storage, network access, or management services.

A camera system is more than an imaging sensor. It may include the sensor itself, an image processor, a lens-control mechanism, firmware, configuration software, a communication interface, and a service that distributes images to other applications. Some systems compress video onboard, attach metadata, perform local analysis, or transmit a live stream over a network. Every additional capability introduces a function that must be understood and maintained.

The sensitivity of camera data depends on the mission. Images may contain people, homes, workplaces, industrial equipment, infrastructure layouts, vehicle identifiers, or details about security arrangements. Even when an image does not directly identify a person, repeated observations may reveal movement patterns or sensitive operational information. Payload security therefore involves confidentiality, integrity, access control, retention, and responsible handling, not merely protection against device malfunction.

Camera interfaces may use high-speed data links, network protocols, serial connections, or vendor-specific control channels. The precise interface depends on the equipment, but the security principle is consistent: a payload should receive only the access needed to perform its role. A camera that provides image frames should not automatically have permission to modify flight-control parameters. A

gimbal controller should not be able to execute unrelated operating-system commands. An imaging application should not be able to read unrelated credentials simply because it runs on the same computer.

Gimbals introduce both mechanical and software considerations. They stabilize a camera or sensor by controlling one or more rotational axes, often using motors, encoders, and an onboard controller. The gimbal may receive commands to change orientation, follow a target, or maintain a particular viewing direction. These commands can influence the quality and direction of the collected data, and a malfunction may create operational concerns if the payload obstructs sensors, affects aircraft balance, or consumes excessive power.

Gimbal-control permissions should be separated from flight-control authority unless a specific, justified integration requires otherwise. A compromised payload application should not be able to change aircraft attitude or disable navigation protections merely because it can command the camera's orientation. The system should define allowed gimbal movement, rate limits where appropriate, and any restrictions needed to prevent interference with other onboard equipment. Where payload movement affects aircraft dynamics or sensor visibility, the integration should be assessed as part of the complete system.

Imaging devices can also introduce network services that are easy to overlook. A payload may provide a configuration page, a live video stream, a discovery protocol, a file-transfer service, or an administrative interface. Default credentials, undocumented services, or excessive network exposure can create pathways into the companion computer or the payload itself. The system inventory should include these interfaces, and the security assessment should determine which ones are required during normal operation, which are needed only for maintenance, and which can be disabled.

Firmware and configuration integrity are equally important. Payload firmware may determine how images are processed, how commands are interpreted, and how data are transmitted. An unauthorized change could affect image availability, alter metadata, weaken access controls, or cause a device to behave unpredictably. Updates should be obtained through approved processes, checked for authenticity where supported, and tested with the aircraft's actual software and hardware configuration. A payload should not be considered secure merely because it was purchased from a reputable supplier.

Payload metadata deserves special care. Images and video may be accompanied by timestamps, coordinates, altitude, orientation, sensor settings, or other information used for mapping and analysis. Such metadata can be as important as the image itself. If it is corrupted or associated with the wrong frame, downstream users may draw incorrect conclusions about where or when an observation occurred. Applications should preserve the relationship between each data record and its supporting measurements, and should identify when the associated navigation information is uncertain.

Data integrity becomes particularly important when payload observations influence autonomous decisions. An object-detection application might identify a landing marker, estimate the location of a target, or classify an area as suitable for a planned action. These outputs should be treated as measurements or recommendations with known limitations, not unquestionable facts. The application should report uncertainty where possible, and the receiving system should apply appropriate validation before using the output for safety-relevant decisions.

Machine-learning models introduce their own lifecycle concerns. A model file may be updated separately from the application that loads it, and changes to model parameters can alter detection behaviour without changing the program's visible interface. The approved model version should therefore be recorded and protected against

unauthorized replacement. Testing should examine representative operating conditions, including lighting variation, motion blur, partial obstruction, sensor noise, and the types of scenes expected during the mission. A model that performs well on a limited set of test images may fail under conditions that differ from its training or validation data.

Payload interfaces should also account for bandwidth and resource constraints. High-resolution imagery can consume substantial storage, processing capacity, and communication bandwidth. If the payload shares a network with telemetry or command traffic, an uncontrolled video stream may compete with information required for safe operation. Network quality-of-service mechanisms, traffic separation, buffering, and rate control can help manage the load. These mechanisms should be validated to ensure that payload activity does not delay safety-relevant messages or exhaust onboard resources.

Data storage should be designed around the mission's needs. Some operations require continuous recording, while others need only selected frames or processed measurements. The system should define how data are stored, encrypted where appropriate, transferred, retained, and deleted. Removable storage media create additional risks because they can be lost, copied, infected, or connected to untrusted computers. Media-handling procedures should identify authorized devices and establish how data integrity is checked after transfer.

A payload may also contain its own wireless interface for configuration or data transfer. If this interface remains active during flight, it can expand the aircraft's exposure beyond the primary command-and-control link. Engineers should determine whether the interface is necessary during operation, whether it can be restricted to a controlled maintenance mode, and how its identity and permissions are managed. Unused wireless capabilities should be

disabled when feasible, particularly if they provide access to sensitive data or configuration functions.

Physical installation matters as much as logical access. Payload connectors can be exposed to vibration, moisture, dust, mechanical stress, and accidental disconnection. An intermittent payload interface may cause repeated service restarts or corrupt data. Power faults can also affect other equipment if the payload draws more current than expected or introduces electrical noise. Proper mounting, connector retention, power budgeting, grounding, and environmental qualification contribute to both payload reliability and system security.

Payload replacement should be treated as a configuration change. A new camera or sensor may use different drivers, network ports, firmware, credentials, power requirements, or data formats. It may also introduce additional services or alter the aircraft's centre of gravity. Before operational use, the organization should verify that the replacement is compatible with the approved architecture, that unnecessary interfaces are disabled, and that the new configuration passes relevant security and safety tests.

# Protecting Application Programming Interfaces and Interprocess Communication

Companion computers frequently use application programming interfaces (APIs) and interprocess communication (IPC) to coordinate software components. These mechanisms allow applications to exchange commands, measurements, events, and status information without requiring every component to communicate directly with every other component. They are

essential for modular software design, but they also establish trust relationships. If those relationships are poorly defined, a weakness in one application can spread into other services or affect the aircraft's mission behaviour.

An API defines how one software component requests a service from another. It may expose operations for retrieving telemetry, requesting a payload action, submitting a mission update, or reading diagnostic information. APIs can be implemented through local sockets, network protocols, message brokers, remote procedure calls, or other communication methods. Regardless of implementation, the interface should define the allowed operations, expected data formats, authentication requirements, error handling, and authorization rules.

A common design weakness is to treat the existence of a local interface as proof that the caller is trustworthy. An API may originally have been designed for use by one application, yet later become accessible to additional processes or network clients. If the service does not authenticate callers or verify their permissions, any process able to reach the interface may be able to invoke operations beyond its intended role. Local communication is not automatically secure, especially on a computer that runs multiple applications or exposes network services.

Each API should follow the principle of least privilege. A caller should receive only the operations and data required for its function. A camera application may need to submit image metadata or request gimbal movement, but it should not need permission to change flight-controller security settings. A reporting service may need read-only telemetry access without the ability to issue commands. Separating read permissions from control permissions reduces the consequences of mistakes and compromise.

Authentication and authorization should be implemented as distinct checks. Authentication determines which application or identity is

making a request, while authorization determines whether that identity may perform the requested action. A system should not grant broad control simply because the caller presents a valid credential. Permissions should be specific to the resource, operation, and context. High-impact actions may require additional validation or an explicit approval step.

Input validation is another fundamental requirement. APIs should verify that incoming data conform to the expected structure, data type, length, range, and format. Numeric fields should be checked for plausible values, coordinate fields should use documented reference systems, and command parameters should be constrained to the supported operating range. Validation should occur at the service boundary rather than relying solely on the calling application to behave correctly.

A well-formed request can still be unsafe. A command may use valid syntax and contain numbers within the permitted range while being inappropriate for the aircraft's current mode or mission state. Services that influence flight-related functions should therefore validate operational context as well as data format. The receiving component should reject requests that conflict with defined safety constraints, regardless of whether the request originated from an approved application.

Replay protection is important for APIs that accept commands or state changes. If a previously valid request can be submitted again without restriction, the system may repeat an action that is no longer appropriate. Session identifiers, sequence numbers, timestamps, request identifiers, and defined expiration rules can help detect duplicate or obsolete requests. These controls must be designed to handle retries and legitimate reconnection behaviour without creating ambiguity about whether an action has already occurred.

Interprocess communication presents similar issues. Processes may exchange information through Unix-domain sockets, shared

memory, message queues, local network sockets, or a publish-and-subscribe messaging system. Each mechanism has different performance and access-control characteristics. Shared memory can provide efficient data exchange but requires careful management of access permissions and synchronization. Message brokers can simplify communication between applications but may become central points of trust if they allow every client to publish or subscribe to every topic.

A messaging architecture should define which applications can publish each category of information and which applications can receive it. Telemetry, payload data, diagnostics, and command requests may have different confidentiality and integrity requirements. If any application can publish messages that resemble authoritative flight commands, the system risks confusing ordinary data with trusted control instructions. Message types, identities, permissions, and validation rules should make these distinctions explicit.

The receiving application should not assume that a message is trustworthy merely because it arrived through the expected broker. A compromised publisher or misconfigured broker may deliver incorrect information through an otherwise legitimate channel. Important messages should be associated with an authenticated source where the architecture supports it, and receiving services should validate their content and context. For safety-relevant requests, the final authority should remain with the component responsible for enforcing the applicable safety rules.

Error handling can also create vulnerabilities. An API that crashes when it receives malformed data may interrupt essential services. A service that returns detailed internal errors to untrusted callers may disclose software paths, configuration details, or implementation information. An application that retries failed requests without limits may exhaust processor resources or flood a service. Robust interfaces should handle invalid requests predictably, limit resource

consumption, record relevant events, and avoid exposing sensitive internal details unnecessarily.

Rate limiting and resource controls can protect services from excessive requests. A camera API, for example, should not allow a client to create unlimited image-processing tasks that exhaust memory. A telemetry broker should not accept an unbounded queue of messages that delays important updates. The limits should reflect the actual operational workload and should not block legitimate time-sensitive communication. Testing should include periods of high payload activity and degraded network conditions.

API versioning and compatibility are also important. Companion-computer software often evolves as applications and libraries are updated. A new application version may interpret a message differently or require fields that older services do not provide. Versioned interfaces, explicit schema definitions, and compatibility tests help prevent a software update from silently changing operational behaviour. When a service receives an unsupported request version, it should fail in a controlled way rather than guessing how to interpret the data.

Secrets must be kept separate from ordinary application configuration wherever practical. API tokens, private keys, passwords, and service credentials should not be embedded directly in source code, exposed in debug logs, or distributed to applications that do not need them. Protected credential storage, restricted file permissions, short-lived tokens where appropriate, and defined revocation procedures reduce exposure. If a credential is compromised, the organization should be able to identify which services it authorizes and revoke it without unnecessarily disabling unrelated functions.

Logging should support both operational diagnosis and security investigation. Records may include the authenticated caller, requested operation, authorization outcome, validation failure,

relevant state transition, and response status. Logs should avoid storing sensitive payload content or credentials unless there is a clear and controlled need. Where the event is safety-relevant, the record should contain enough information to reconstruct the sequence of events while respecting data-minimization requirements.

Testing should examine both expected use and misuse. Engineers should verify that unauthorized clients cannot access restricted operations, malformed requests are rejected, expired commands are not accepted, and resource limits behave as intended. They should also test how the interface behaves during service restart, network interruption, application crash, and recovery. These tests should be performed in an authorized development or simulation environment and should avoid interfering with live aircraft operations.

## Separating Payload Permissions From Flight-Critical Functions

The separation of payload permissions from flight-critical functions is a foundational design principle for secure unmanned aircraft. A payload application may process images, record environmental measurements, or manage a specialized instrument. A flight controller must maintain stable control, interpret critical sensor data, and enforce the aircraft's safety constraints. When these responsibilities are mixed without clear boundaries, a failure or compromise in a comparatively low-criticality application can affect functions whose failure may lead to loss of the aircraft or harm to people and property.

Separation begins with an explicit allocation of authority. The system architecture should identify which components may observe aircraft state, which may request mission changes, which may control payload devices, and which may modify flight-critical configuration.

The design should state which component makes the final decision for each action. Without this clarity, two services may believe they have authority over the same function, or a payload application may gain control simply because its software interface permits it.

A useful arrangement allows payload software to request limited actions while leaving final validation to the flight-control system or another designated safety component. A vision application might propose a landing position based on an observed marker. The flight-control system can evaluate whether the position is valid, whether navigation confidence is sufficient, whether the aircraft is in the correct mode, and whether the requested movement remains within approved limits. This arrangement allows advanced onboard perception without granting the perception software unrestricted control.

The communication interface should enforce this division. Payload applications should not have direct access to low-level actuator outputs, motor-control registers, or protected flight-controller configuration unless a specific design requires such access and the resulting risk has been assessed. In ordinary architectures, these functions should remain under the authority of the flight controller. A companion application can send a high-level request through a defined interface, but the receiving system should decide whether the request is permitted.

Hardware separation can strengthen software boundaries. Where feasible, flight-critical control and payload processing should use separate processors, power domains, or communication interfaces appropriate to their functions. Separate hardware does not remove all common-mode risks, because the components may still share power distribution, network infrastructure, or physical conditions. Nevertheless, careful separation can reduce the chance that a resource-exhaustion event or software compromise in the payload system directly disrupts the flight-control processor.

When separate hardware is impractical, strong software isolation becomes more important. Applications should use separate operating-system identities, restricted access to devices, and controlled communication channels. The flight-control interface should expose only the functions that companion applications require. The operating system should not allow an ordinary payload process to modify protected files or gain administrative privileges through an unnecessary service. Isolation must be validated through testing rather than assumed from the presence of different process names or directories.

Safety-critical authority should also be protected against accidental configuration changes. A payload installation tool should not silently alter the flight controller's failsafe settings, navigation-source selection, or command permissions. Configuration changes affecting flight behaviour should pass through a documented approval process and should be associated with an identifiable software or hardware configuration. If a payload requires a special operating mode, the system should verify the required conditions before enabling it and should restore the appropriate configuration when the payload is removed or the mission ends.

The design should consider failure propagation. A camera application may crash repeatedly, a machine-learning model may consume excessive memory, or a payload service may flood the local network with data. These faults should not prevent the flight controller from maintaining stable control. Resource limits, process supervision, network traffic management, and isolation between services help contain such failures. Where a payload is essential to the mission, the aircraft should define what happens when it becomes unavailable. Where the payload is nonessential, the aircraft should generally be able to continue safe flight without it, subject to the approved operating concept.

The system should distinguish between a payload failure and a flight-control failure. A failed camera may make a particular inspection task impossible, but it should not automatically require the aircraft to lose control or terminate the flight in an unsafe manner. Conversely, if the aircraft relies on a camera for navigation or landing, loss of that camera may affect a safety-relevant function. The architecture and operational procedures must identify such dependencies accurately. A device's classification as a payload does not determine its criticality; its actual role in the system does.

Permission separation should extend to remote operators and maintenance personnel. An individual authorized to retrieve imagery should not automatically be permitted to alter flight-control settings. A technician configuring a gimbal should not necessarily be able to initiate a flight. Role-based access control can reflect these differences, while additional approval may be appropriate for actions with significant safety consequences. Permissions should be reviewed when personnel responsibilities change, and unused accounts should be disabled.

Secure boot, signed firmware, and controlled updates can help protect both sides of the boundary. The flight controller should verify the integrity of its approved software where supported, while the companion computer should protect its operating system and applications through suitable update and access controls. Payload firmware should be included in configuration management when it can influence mission behaviour or data integrity. A secure flight controller cannot compensate for a compromised companion application if the interface grants that application excessive authority.

The system should also defend against incorrect but authenticated requests. A payload application may be legitimately signed and still contain a software defect. It may misinterpret an image, use stale navigation data, or produce a target outside the expected operating

region. Authentication establishes the identity of the application; it does not establish that every decision it makes is correct. The receiving component should therefore enforce independent limits and reject requests that conflict with the aircraft's state or approved mission constraints.

Independent monitoring can strengthen the boundary further. The flight controller may monitor command frequency, command validity, source identity, and changes requested through the companion interface. Unexpected request patterns can trigger a warning, a restriction of the relevant interface, or another defined response. Such monitoring must be carefully designed to avoid rejecting legitimate commands during unusual but valid operations. It should be based on the system's documented command model and validated through controlled testing.

A secure architecture should also define the behaviour required during loss of communication between the companion computer and flight controller. If the companion computer stops sending mission updates, the flight controller should not interpret silence as permission to continue an unbounded sequence of previously requested actions. It should follow the approved policy for stale requests and missing status information. If the aircraft can continue safely using onboard control and a validated mission plan, it may do so within the defined limits. If the missing companion function is necessary for safe operation, the system should enter the appropriate contingency.

Recovery from a companion-computer failure should preserve the separation of authority. Restarting a failed application should not automatically restore privileges that were deliberately revoked or accept commands left over from the previous session. The system should establish a fresh communication state, verify the identity of the restored service where supported, and reject expired requests. If the companion computer has been replaced or reimaged, its identity

and approved configuration should be verified before it is allowed to interact with flight-critical functions.

Security and safety testing should confirm that the separation works under realistic conditions. Tests can examine whether payload applications are prevented from accessing protected configuration, whether excessive image processing affects flight-control timing, whether invalid mission requests are rejected, and whether loss of the companion computer triggers the expected response. Hardware-in-the-loop and software-in-the-loop environments can help reproduce faults without placing an aircraft at unnecessary risk. Results should be documented against defined acceptance criteria, including both successful operation and failure containment.

The final architecture should make the limits of each component explicit. Payload systems should collect and process mission information within their assigned permissions. Companion computers should coordinate applications and submit requests through controlled interfaces. Flight-critical components should retain authority over essential control and safety constraints. When those boundaries are clear, the aircraft becomes easier to test, maintain, and secure because a defect in one component is less likely to spread into unrelated functions. This separation also provides a sound basis for adding new payloads and autonomous capabilities without granting each new application unnecessary influence over the aircraft's fundamental behaviour.

# Physical Interfaces and Hardware Security

## Debug Ports, Test Points, and Embedded Development Interfaces

Physical interfaces are among the most important security considerations in an unmanned aircraft because they connect electronic components to the tools and systems used to develop, manufacture, configure, test, and maintain them. Although these interfaces are essential for engineering work, they can also create opportunities for unauthorized access when they remain exposed, insufficiently protected, or accessible without appropriate controls. A drone may employ sophisticated encryption for its wireless communications and still have weaknesses in its physical design if an individual can connect directly to an internal computing component and obtain access to information or functions that should remain restricted.

A modern drone typically contains several interconnected electronic subsystems. The flight controller processes sensor measurements and executes flight-control algorithms. A companion computer may handle computer vision, mission planning, data processing, or communication with external services. Additional circuit boards manage power distribution, motor control, navigation, cameras, and other payload equipment. Depending on the platform, these components may contain microcontrollers, application processors, programmable logic, dedicated communication controllers, and nonvolatile memory. Each component can expose different interfaces for development, manufacturing, diagnostics, and maintenance.

The presence of a physical interface does not automatically represent a vulnerability. Engineering teams require reliable ways to program processors, inspect system behavior, verify circuit operation, and diagnose faults. A debug connector can substantially reduce maintenance time, while carefully designed test points can make it possible to measure voltages or investigate intermittent electrical problems without replacing an entire board. The security question is whether these capabilities remain available only to authorized personnel under controlled conditions or whether they create an unnecessary route into the system.

Debug ports are commonly associated with processor-level development. Depending on the processor and its configuration, a debug interface may permit an engineer to inspect execution state, examine registers, pause a running program, or access selected memory regions. Such functions are valuable when diagnosing a defective flight-control algorithm or investigating a processor that unexpectedly resets. However, unrestricted debugging can weaken the separation between ordinary software permissions and privileged system access. An individual with sufficient physical access and the appropriate equipment may be able to investigate internal operations in ways that would not be possible through the drone's normal user interface.

The security implications depend on the hardware architecture, the processor's security configuration, and the controls implemented by the manufacturer. Some devices permit extensive debugging during development but support permanent or controlled restrictions in production. Others provide authentication mechanisms, lifecycle states, or access limitations that reduce the capabilities available after deployment. A secure design must account for these differences rather than assuming that every interface behaves identically.

Test points present a related but distinct concern. These are conductive pads, vias, or designated measurement locations placed

on a printed circuit board to support manufacturing tests and technical investigations. They may expose power rails, communication signals, reset lines, clock signals, or other electrical connections. Some are clearly marked, while others are difficult to distinguish from ordinary circuit features. Their existence is normal in electronic design, and their presence alone does not establish that sensitive functionality can be accessed.

The security risk arises when a test point exposes a sensitive signal or provides an unintended route to a privileged interface. A reset line, for example, may be relevant to how a device starts and how its security checks are applied. A serial communication signal may provide diagnostic output containing operational information. A measurement point connected to a memory device might reveal activity that helps an authorized assessor understand the system's design. Whether any of these conditions represents an exploitable weakness depends on the complete circuit and the protections surrounding it.

Embedded development interfaces extend beyond debugging. During manufacturing, devices may require programming connections, calibration facilities, boot-mode selection, and mechanisms for installing initial firmware. These functions can be distributed across multiple connectors or integrated into a single interface. If manufacturing privileges remain available without suitable restrictions, a deployed drone may retain capabilities intended only for production facilities. This can undermine the security assumptions on which the rest of the system depends.

A disciplined hardware-security assessment therefore begins with an inventory of interfaces and their intended purposes. Engineers identify the processor, memory devices, communication controllers, and supporting circuits, then consult approved schematics, board documentation, component datasheets, and manufacturing records. The purpose is to establish which interfaces exist, what functions

they provide, and which of those functions should remain accessible during normal operation.

The assessment should distinguish between interfaces that are physically present, interfaces that are electrically connected, and interfaces that are functionally enabled. A connector may be unpopulated, a test pad may connect only to a harmless measurement point, or a debug facility may be disabled by the processor's security configuration. Conversely, a small, undocumented connection may lead to a function with significant privileges. Visual inspection alone cannot reliably establish the security significance of every interface.

Production hardware should follow a deliberate interface policy. Development boards may require broad diagnostic access, whereas aircraft intended for routine operational use should expose only the capabilities needed for authorized maintenance. Unnecessary connectors can be omitted, sensitive test points can be removed or restricted, and interfaces that must remain can be protected through suitable authentication, access controls, enclosure design, or processor configuration. These decisions should be made during hardware development rather than added as an afterthought.

Maintenance procedures are equally important. A service technician may legitimately need access to an internal interface to diagnose a fault, update approved software, or verify a hardware replacement. That access should be governed by documented procedures, appropriate authorization, and controls that prevent diagnostic work from silently weakening the system's security configuration. Where practical, maintenance actions should be recorded so that investigators can determine what was accessed, which changes were made, and whether the equipment returned to an approved state.

Physical interfaces must also be evaluated in relation to the drone's operating environment. Aircraft used in industrial inspection, agriculture, emergency response, surveying, or remote infrastructure

monitoring may be transported frequently, stored in different locations, and serviced by multiple organizations. This creates opportunities for accidental damage, unauthorized handling, and inconsistent maintenance practices. A security control that works in a tightly managed laboratory may be inadequate for equipment that spends much of its service life in uncontrolled environments.

The objective is not to eliminate every physical interface. It is to ensure that each interface has a justified purpose, an understood security impact, and an appropriate level of protection throughout the device's lifecycle. A well-designed system preserves the diagnostic capabilities engineers need while preventing those same capabilities from becoming an overlooked path to sensitive flight-control functions, protected data, or privileged configuration.

## UART, SPI, I²C, JTAG, and SWD in Drone Electronics

Serial communication and processor-debugging interfaces are widely used in embedded electronics because they provide practical ways to exchange data, connect components, and support software development. In drone systems, these interfaces may connect the flight controller to sensors, memory devices, radio modules, power-management circuits, and companion computers. Understanding their characteristics helps engineers identify which connections require protection and which are necessary for reliable operation.

Universal asynchronous receiver-transmitter communication, commonly abbreviated UART, is a method of serial data exchange frequently used for console access, telemetry connections, and communication between embedded components. A typical UART connection uses separate transmit and receive signals and a shared reference ground. Depending on the implementation, additional

signals may support flow control or other functions. UART itself does not provide encryption, authentication, or authorization. Those protections must be implemented by the software protocol, surrounding hardware, or both.

A UART interface can serve several legitimate purposes. During development, a console may provide diagnostic messages that reveal boot progress, sensor initialization failures, or software exceptions. A maintenance interface may allow an authorized technician to inspect system health or perform a supported recovery procedure. Communication between internal modules may also rely on UART when the required data rate and wiring simplicity make it suitable.

Security depends heavily on what is exposed through the connection. A read-only diagnostic console that reports non-sensitive status information has a different risk profile from a console that provides an administrative shell or accepts unrestricted configuration commands. Even a console that appears harmless can disclose internal details through boot messages, software versions, error traces, or network configuration information. Such information may help an assessor understand the system, but it can also increase the risk of subsequent attacks if made available to unauthorized individuals.

A secure UART implementation should therefore restrict unnecessary functionality, avoid transmitting secrets in diagnostic messages, and prevent privileged command execution without proper authorization. If a console is required in production, access should be controlled according to its purpose and operational context. Disabling unused console access, limiting available commands, and ensuring that recovery modes enforce appropriate security checks can reduce unnecessary exposure.

Serial Peripheral Interface, commonly called SPI, is a synchronous communication interface frequently used between processors and peripheral devices. It typically employs a clock signal, a controller-to-

peripheral data line, a peripheral-to-controller data line, and one or more selection signals. SPI is often chosen for flash memory, displays, radio components, and sensors because it can provide high data rates with relatively straightforward hardware.

SPI does not inherently authenticate connected devices or encrypt transferred data. Its security properties depend on the connected components, the surrounding circuit, and the software controlling the exchange. If an SPI-connected memory device contains firmware, configuration data, calibration information, or other sensitive material, the design must consider what an individual with physical access to the board could learn or alter. The exact risk varies according to the memory technology, circuit topology, processor configuration, and any protection features supported by the device.

For example, an external flash memory component may store executable firmware or persistent settings. If the system relies exclusively on the assumption that the memory chip is physically inaccessible, the security design may be fragile when the board can be removed from its enclosure. Stronger designs verify the authenticity of executable code before allowing it to run and protect sensitive configuration data against unauthorized modification. Where confidentiality matters, the system may also require encryption and careful management of the keys needed to decrypt stored information.

Inter-integrated circuit communication, commonly abbreviated I²C, is another synchronous interface widely used for communication between processors and lower-speed peripherals. It generally uses two signal lines: a serial data line and a serial clock line. Multiple compatible devices can share the same bus, with addressing and bus-management rules coordinating communication.

In drone electronics, I²C may connect environmental sensors, power monitors, configuration devices, and other peripherals. The shared-bus structure can simplify board design, but it also means that the

security and reliability of individual devices can affect the broader system. An unexpected peripheral response, electrical fault, or malfunctioning device may interfere with communication involving other components on the bus.

I²C does not inherently guarantee that a connected peripheral is genuine or authorized. A device that responds at an expected address is not necessarily the component the system intended to communicate with. For security-sensitive applications, the surrounding software may need to validate device behavior, check expected identities where supported, and detect unexpected changes in configuration or measurement patterns. Not every I²C peripheral supports cryptographic authentication, so the required controls must be selected according to the hardware's actual capabilities.

The shared nature of the bus also makes fault handling important. A peripheral that holds a signal in an invalid state may prevent communication with other devices, potentially affecting sensor availability or power monitoring. Engineers should consider how the flight controller behaves when a peripheral stops responding, returns implausible data, or repeatedly causes communication errors. A robust implementation recognizes these conditions, reports them appropriately, and enters a safe operational state when necessary.

Joint Test Action Group, commonly known as JTAG, refers to a family of standards and techniques originally developed for testing and verifying digital electronic assemblies. JTAG is widely associated with boundary-scan testing, which allows compatible components and circuit connections to be examined without requiring direct physical access to every individual connection. It can also support processor debugging when the device and its configuration permit that functionality.

In embedded systems, a JTAG connection may provide access to powerful diagnostic capabilities. Depending on the target processor and enabled features, these may include examining execution state,

controlling program execution, or interacting with memory. Such capabilities make JTAG valuable during development and fault investigation, but they also require careful management on deployed equipment. A security assessment must establish whether JTAG is present, whether it remains enabled, and what restrictions apply in the device's production configuration.

Serial Wire Debug, or SWD, is a debugging interface used by certain microcontroller architectures. It commonly provides a clock signal and a bidirectional data signal, reducing the number of physical connections required for debugging. Its smaller pin count can be advantageous in compact flight controllers, where board space and connector availability are limited.

SWD is not automatically more or less secure than JTAG simply because it uses fewer signal lines. The relevant considerations are the processor's debug architecture, security settings, access-control mechanisms, and lifecycle configuration. A device may support debug authentication, restrict access to sensitive memory, or disable selected debug functions after production provisioning. Another device may leave broad debugging capabilities available. Security teams must assess the actual implementation rather than relying on the name of the interface.

The distinction between communication interfaces and debug interfaces is important. UART, SPI, and I²C are primarily used to exchange operational data between components, although UART may also provide a software console. JTAG and SWD are primarily associated with testing and debugging, though their exact capabilities vary by device. In practice, a single board may contain all five interfaces, and their security significance may overlap. A serial console could provide administrative access, while a debug interface could reveal why a security control behaves incorrectly. An external memory bus could expose firmware storage, while a sensor bus could affect the integrity of measurements used during flight.

An effective assessment evaluates each interface in its surrounding context. Engineers should identify the connected components, document the expected communication patterns, determine whether sensitive information crosses the interface, and establish what happens when communication fails. The assessment should also consider whether access to one interface could undermine protections associated with another. For example, disabling a debug port offers limited assurance if a separate manufacturing interface still permits unauthorized changes to the device's security configuration.

Testing should take place on equipment that the assessor owns or has explicit permission to examine. A controlled bench environment allows electrical behavior and interface functionality to be evaluated without placing an aircraft, personnel, or surrounding property at risk. Before connecting test equipment, engineers should confirm voltage levels, signal characteristics, grounding requirements, and the manufacturer's approved procedures. Incorrect connections can damage components or produce misleading results.

The purpose of examining these interfaces is to establish whether the hardware and firmware enforce their intended security boundaries. The mere discovery of a serial connection is not evidence of a vulnerability. The important questions concern the privileges it provides, the information it exposes, the commands it accepts, and the effect of unauthorized access on the confidentiality, integrity, availability, and safe operation of the drone.

# Memory Protection, Device Identity, and Hardware Security Features

Memory protection is a central component of embedded security because drone electronics frequently store executable software, calibration values, operational settings, credentials, and mission-related information. These assets differ in sensitivity, but each must be protected against the forms of access or modification that could compromise system reliability. A flight controller that executes altered firmware may behave unpredictably, while a device that loses critical calibration values may produce inaccurate measurements even when its software remains authentic.

Embedded memory generally falls into several categories. Volatile memory, such as static or dynamic random-access memory, stores information while the system is powered. Nonvolatile memory, including flash and electrically erasable programmable read-only memory, can retain information after power is removed. Some microcontrollers include internal flash and RAM, while more complex platforms use external memory components. The choice of memory architecture affects performance, cost, storage capacity, and the security controls available to the designer.

Memory protection begins with a clear understanding of which information resides in each location. Executable firmware may require integrity and authenticity checks. Cryptographic keys may require both confidentiality and protection against unauthorized replacement. Calibration parameters may require integrity and controlled update procedures. Temporary operational data may need to be cleared when no longer required. Treating every memory region as equally sensitive can lead to unnecessary complexity, while failing to identify critical assets can leave important information inadequately protected.

Many microcontrollers provide hardware features intended to restrict access to memory regions or privileged functions. Depending on the processor, these features may include memory protection units, execution permissions, readout protection, secure and non-

secure execution regions, or restrictions on access to particular peripherals. These mechanisms can reduce the consequences of a compromised software component by limiting what it can read, modify, or execute.

A memory protection unit, where supported, can establish rules governing access to defined memory regions. For example, a system may distinguish between code that can execute and data that can be read or written, or restrict particular regions to privileged software. These controls help enforce separation between components that should not have unrestricted access to one another. Their effectiveness depends on correct configuration and on the processor architecture's actual capabilities.

Hardware-enforced isolation is particularly valuable in complex flight systems. A flight controller may run control algorithms, communication services, logging functions, and maintenance routines. If every component operates with unrestricted access to all memory and peripherals, a defect in one component may have consequences far beyond its intended responsibilities. Appropriate isolation can reduce the impact of software errors and limit the authority available to less-trusted functions.

Readout protection can restrict direct access to certain internal memory contents through supported programming or debugging mechanisms. However, readout protection is not synonymous with complete device security. Its strength depends on the implementation, the selected protection level, the available recovery procedures, and whether other interfaces or privileged software functions can expose the same information. Engineers should consult the component documentation and verify the configured protection state rather than assuming that enabling a feature provides comprehensive confidentiality.

Secure boot is another important hardware-supported security function. In a typical secure-boot design, a trusted initial component

verifies the authenticity or integrity of the next software component before transferring control to it. The process may continue through additional boot stages until the system reaches its operational state. The initial trust anchor must be protected, and the verification process must reject software that does not satisfy the defined policy.

Secure boot is most effective when it verifies authenticity using an appropriate cryptographic mechanism and a protected trust anchor. A simple checksum can detect some accidental corruption, but it does not necessarily prevent an attacker from creating modified firmware with a matching checksum. Cryptographic signatures, when correctly implemented and validated against a trusted public key or equivalent verification mechanism, provide stronger evidence that the firmware was approved by an authorized signing process.

A secure-boot design must also consider rollback attacks. Even if a firmware image was legitimately signed, an older version may contain a vulnerability that has since been corrected. If the system accepts any correctly signed version, an unauthorized party who can install old software may be able to reintroduce a known weakness. Where appropriate, version policies or protected monotonic counters can help prevent the installation of software that falls below an approved security level.

Secure boot does not automatically make a running system secure. Once approved firmware begins executing, software vulnerabilities, incorrect access controls, insecure communication protocols, and unsafe configuration decisions can still create risks. Secure boot establishes an important foundation by limiting which software is permitted to start, but it must be combined with runtime protections, secure update mechanisms, and sound application design.

Device identity is another important consideration. A drone may communicate with a ground station, a maintenance tool, a cloud service, or another onboard component. These systems need reliable

ways to establish which device is communicating and whether that device should be permitted to perform a particular action. An identifier printed on a label or stored in a configuration file may help distinguish devices operationally, but it does not necessarily prove that the communicating device possesses an authentic identity.

Cryptographic device identity generally relies on credentials that can be verified by an authorized counterpart. Depending on the design, a device may possess a private key and a corresponding certificate or other trusted identity record. The private key must remain under the device's control, while the verification process must establish that the associated identity is recognized and permitted for the requested operation.

Hardware-backed key storage can reduce the risk of exposing private keys through ordinary software access. Some processors provide secure key-storage regions, cryptographic accelerators, or isolated execution environments. Other designs use dedicated secure elements or trusted platform components to perform cryptographic operations without making the underlying private key directly readable by the main application processor. The precise capabilities vary significantly between products.

A secure element can provide a protected environment for storing credentials and performing selected cryptographic functions. Its value depends on how it is integrated into the wider system. If the surrounding software permits an unauthorized application to request unrestricted signing operations, protected key storage alone may not prevent misuse. Access to cryptographic functions must therefore be governed by suitable authorization and policy controls.

Device identity must also account for the provisioning process. A cryptographic key that is generated securely can still be mishandled if it is copied into an ordinary configuration file, included in a software image, or reused across an entire fleet. Shared secrets may be difficult to revoke selectively because the compromise of one

device can affect others. Where practical, individual device credentials and controlled provisioning procedures provide a stronger foundation for fleet security.

Hardware security features should be selected according to the threat model, operational requirements, and consequences of compromise. A small research drone may have different cost and maintenance constraints from an aircraft used to inspect critical infrastructure. Nevertheless, every platform benefits from understanding which security functions the hardware supports, how they are configured, and what happens when an attacker attempts to bypass or misuse them.

## Secure Storage, Cryptographic Secrets, and Tamper Resistance

Secure storage concerns the protection of information that must remain confidential, authentic, or available throughout a device's operational life. In drone systems, this information may include cryptographic keys, authentication credentials, wireless-network settings, mission parameters, calibration data, maintenance records, and sensitive operational logs. The appropriate protection depends on the purpose of the information and the consequences of its disclosure or alteration.

Cryptographic secrets require particular care because their compromise can undermine controls across multiple subsystems. A private key used to authenticate a device may allow an unauthorized party to impersonate that device if the key is copied and accepted by other systems. A shared secret used across a fleet can create a wider exposure if one aircraft is lost or physically compromised. A key used to protect stored information may also become ineffective if it is

saved beside the encrypted data without adequate separation or access controls.

Encryption is useful only when key management is sound. Protecting a file with a strong cryptographic algorithm does not provide meaningful confidentiality if the decryption key is stored in the same accessible location or can be recovered through an unrestricted diagnostic interface. Engineers must determine where keys are generated, how they are stored, which components can use them, and what procedures govern rotation, revocation, replacement, and recovery.

Whenever feasible, device-specific credentials should be generated or provisioned through a controlled process. Private keys should not be embedded in publicly distributed firmware images or copied indiscriminately between devices. Manufacturing systems should limit access to provisioning material, record relevant operations, and prevent production credentials from being exposed through ordinary development tools. The resulting process should make it possible to associate a device with its authorized identity without making sensitive key material broadly available.

A secure element or equivalent hardware-backed mechanism can help protect keys from direct extraction, but it should not be treated as an absolute guarantee. Depending on the component, the hardware may support protected storage, cryptographic operations, access policies, or resistance to selected physical attacks. These protections have defined limits, and their effectiveness depends on the implementation and the attacker's capabilities. A security design should rely on documented features and validated behavior rather than broad marketing claims.

Secure storage also requires integrity protection. Some configuration values may not be confidential, yet unauthorized changes could affect aircraft behavior. Calibration parameters, sensor settings, safety thresholds, and trusted software-version records may need

integrity checks even if their contents are not secret. A secure system validates such values when they are loaded and ensures that changes can occur only through authorized processes.

Where stored data is encrypted, the system must manage initialization, updates, and recovery carefully. Encryption can protect information against disclosure when storage is removed or copied, but it does not necessarily prevent an attacker from deleting the data, replacing it with invalid content, or repeatedly triggering failures. Integrity protection, authenticated encryption where appropriate, reliable error handling, and protected recovery procedures address different aspects of the problem.

Cryptographic implementation should favor established algorithms and well-reviewed libraries appropriate to the processor and operational requirements. Designing a proprietary encryption method is rarely justified, particularly when the device must be maintained for years and assessed by different engineering teams. The implementation must also consider random-number generation, key lifetime, algorithm selection, and the possibility that sensitive material could appear in debug output, crash dumps, or temporary memory.

Secure deletion presents a further challenge in embedded systems. Removing a file or clearing a configuration entry does not necessarily erase every physical copy of the underlying data. Flash storage may use wear-leveling, remapping, or other mechanisms that make the relationship between logical addresses and physical storage locations more complex. For highly sensitive information, the system should use storage technologies and sanitization procedures suited to the required assurance level. Where supported, cryptographic erasure may be useful if it securely renders the relevant encrypted data inaccessible by destroying the necessary key material, but the approach depends on correct key management and implementation.

Tamper resistance concerns the ability of a device to withstand, detect, or limit the consequences of unauthorized physical interference. It should be distinguished from tamper evidence, which helps reveal that interference may have occurred. A sealed enclosure may discourage casual access or provide visible evidence of opening, but it does not necessarily prevent a determined person from reaching the electronics. Likewise, a component marketed as tamper-resistant may offer protection only against a defined range of physical threats.

The appropriate level of tamper protection depends on the drone's environment and the value of the information it carries. An aircraft used in a controlled laboratory may need only basic enclosure protection and restricted maintenance access. A system deployed in remote locations may require stronger protection because physical custody cannot be guaranteed. Equipment carrying sensitive credentials or operating in a high-consequence environment may need a more extensive combination of hardware security, monitoring, and operational safeguards.

Mechanical design is an important part of this protection. Enclosures can reduce exposure to dust, moisture, accidental contact, and casual interference. Fasteners, internal partitions, connector placement, and board mounting can make unauthorized access more difficult. These measures must remain compatible with cooling, vibration tolerance, maintenance requirements, and safe battery handling. A design that protects electronics physically but traps excessive heat or makes emergency maintenance impractical may introduce different operational risks.

Tamper-evident seals can support maintenance accountability when their limitations are understood. A damaged or missing seal may indicate that an enclosure was opened, but it does not prove that a malicious act occurred, and an intact seal cannot establish that every internal component remains trustworthy. Seals should be treated as

one source of evidence alongside maintenance records, configuration checks, and system-integrity verification.

Some hardware platforms support tamper-detection mechanisms that respond to defined events, such as enclosure opening or unauthorized changes to security-critical state. Depending on the design, a response may include recording an event, restricting access to sensitive operations, or requiring reauthorization. Destructive responses that erase keys or permanently disable equipment require particular caution because false alarms, component faults, or legitimate maintenance could make the system unusable. Any response should be proportionate to the threat and compatible with safe aircraft operation.

Environmental conditions also influence physical security. Vibration, temperature variation, humidity, corrosion, and repeated transport can weaken connectors or damage protective features. An intermittent connection may resemble deliberate interference even when the underlying cause is ordinary wear. Security monitoring should therefore distinguish, where possible, between verified tampering, hardware faults, and uncertain anomalies. Accurate event records help maintenance personnel investigate without drawing unsupported conclusions.

The relationship between secure storage and tamper resistance is strongest when both are incorporated into a broader lifecycle strategy. Hardware controls can reduce direct access to sensitive assets, while cryptographic protections limit the usefulness of copied storage. Controlled provisioning can prevent unauthorized devices from obtaining trusted identities, and revocation procedures can limit the consequences when a device is lost or suspected of compromise. No single mechanism addresses every threat, but a layered design can make successful compromise more difficult and reduce its operational consequences.

# Conducting Non-Destructive Hardware Assessments on Authorized Equipment

A non-destructive hardware assessment examines a device's physical security properties without intentionally damaging components, permanently altering protective features, or placing the equipment under unnecessary stress. In drone cybersecurity, this approach is valuable because flight controllers, communication modules, navigation components, and power-management systems may be expensive, difficult to replace, or essential to ongoing operations. A careful assessment can produce meaningful security findings while preserving the equipment for subsequent testing and maintenance.

Authorization is the first requirement. The assessor should establish ownership or obtain explicit permission from the responsible organization before examining the hardware. The agreed scope should identify the specific aircraft, circuit boards, peripherals, interfaces, and software versions included in the assessment. It should also state which activities are permitted, which are excluded, how findings will be documented, and who is responsible for approving changes. Permission to inspect a device does not automatically imply permission to bypass protections, extract sensitive credentials, alter firmware, or conduct flight tests.

The assessment should begin with a review of available documentation. Relevant materials may include circuit diagrams, board layouts, component datasheets, service manuals, firmware configuration records, manufacturing instructions, and previous maintenance reports. Documentation helps the assessor distinguish intended interfaces from accidental damage, understand the purpose of security controls, and identify the conditions under which particular functions should be enabled.

Where documentation is incomplete, the assessor should record uncertainty rather than assume a particular connection or function. Board revisions may differ, components may have been substituted during manufacturing, and connectors that look similar may serve different purposes. The exact component marking and revision should be verified whenever possible. If a critical detail cannot be established safely, the assessment should identify it as an unresolved question requiring manufacturer clarification or a more specialized investigation.

A baseline record provides a reference for later findings. This may include the device model, hardware revision, visible serial or asset identifier, firmware version, enclosure condition, available security configuration, and known maintenance history. Sensitive identifiers should be handled according to the organization's information-protection requirements. Photographs can document connector placement, seal condition, board layout, and other relevant observations, provided the images do not unnecessarily expose credentials, private operational data, or restricted design information.

Before opening an enclosure, the assessor should review the manufacturer's maintenance procedures and evaluate electrical and mechanical hazards. Lithium-based batteries can deliver substantial current and present fire risks if damaged or short-circuited. Capacitors may retain charge after power is removed, and delicate connectors can be damaged by incorrect handling. Electrostatic discharge can harm semiconductor components without producing visible evidence. Suitable precautions may include disconnecting power as instructed, using appropriate electrostatic-discharge controls, securing loose parts, and working in a clean, well-lit environment.

The assessor should not open sealed or safety-critical equipment unless the scope and maintenance documentation permit it. Some

enclosures provide environmental protection or contain components that require specialized handling. If opening the device would compromise a warranty, invalidate a safety certification, damage a protective seal, or create a risk that cannot be adequately controlled, external inspection and documentation review may be the appropriate limit of the assessment.

An external inspection can reveal important information without disturbing the hardware. Engineers can examine enclosure integrity, connector accessibility, visible signs of wear, exposed conductors, labels, fasteners, and the condition of protective seals. They can also identify whether external ports are adequately protected from accidental connection and whether the enclosure design supports the intended operating environment. Findings should distinguish observable facts from assumptions about what an unknown interface might permit.

When internal inspection is authorized, the next step is to map the relevant hardware at a level appropriate to the assessment. The goal is to identify processors, memory devices, communication controllers, secure elements, debug connectors, and other components associated with security-sensitive functions. Component datasheets and approved board documentation help establish whether the hardware supports features such as secure boot, memory access restrictions, cryptographic operations, or controlled debugging.

Visual inspection should not be used as a substitute for configuration verification. A microcontroller may support readout protection, but its actual protection state may not be apparent from the circuit board. A processor may include secure-boot capabilities that have not been enabled. Conversely, a board may lack an obvious debug connector while still exposing an interface through another accessible connection. The assessor should record which security properties are confirmed, which are inferred from documentation, and which remain unverified.

Non-destructive electrical measurements can help establish the behavior of approved test points and power interfaces. Such measurements should be limited to documented or otherwise safely identified locations, using instruments and methods appropriate to the circuit. Engineers should confirm expected voltage ranges, grounding arrangements, and signal characteristics before connecting probes. An unknown pad should not be treated as safe simply because it is small or located near a familiar connector.

Where serial or debugging interfaces are within scope, their presence and intended purpose should be assessed before any connection is attempted. The assessor should verify the required electrical levels and approved access method, then determine whether the interface is expected to be active in production hardware. Testing should use documented, read-only procedures where possible. If access would require changing irreversible security settings, defeating a protection mechanism, or risking the loss of stored data, the assessor should obtain explicit additional authorization and evaluate whether a safer alternative can answer the security question.

A sound assessment places particular emphasis on observable security properties. For example, the assessor may determine whether production documentation requires debugging to be restricted, whether the hardware includes a supported authentication mechanism, and whether maintenance procedures verify the expected protection state. If an approved test confirms that an interface provides access beyond its documented role, the finding should describe the observed behavior, the affected asset, and the potential consequence without exaggerating what the evidence demonstrates.

Secure-boot validation requires similar discipline. The assessor should first establish the expected boot-verification policy and identify which component is responsible for enforcing it. If the organization has an approved test procedure and a designated test

unit, validation may involve confirming that the system accepts authorized firmware and responds appropriately to a controlled, known-invalid test image. Such testing should not be performed casually on an operational aircraft. It may render the device unbootable, invalidate configuration, or require a recovery procedure that is not available in the field.

Where a safe negative test is not authorized or practical, the assessor can examine configuration evidence, documented provisioning records, supported status reports, and the design of the boot-verification process. These sources may not provide the same level of assurance as a carefully controlled functional test, but they can still reveal gaps in the verification process. The report should clearly state the level of evidence obtained rather than claiming that a feature has been fully validated when only its documented presence has been confirmed.

Memory and storage assessments should focus on the protections appropriate to the information involved. The assessor can identify where firmware and persistent configuration are stored, determine whether sensitive credentials are expected to reside there, and review whether the design uses hardware-backed protection or encryption where needed. Any attempt to access protected information should remain within the approved scope. Extracting private keys or personal data is not necessary merely to establish that a storage-protection mechanism exists, and unnecessary collection of sensitive information increases both risk and reporting obligations.

Device-identity controls can be assessed by reviewing the provisioning process, the identity-verification mechanism, and the procedure for revoking credentials when a unit is lost or decommissioned. The assessor should establish whether individual devices receive distinct credentials where required, whether private keys are protected against ordinary software access, and whether authorized services validate identities before accepting privileged operations. If the evidence does not establish how a credential is

protected internally, that uncertainty should be documented rather than resolved through an unapproved extraction attempt.

A non-destructive assessment should also consider the integrity of physical maintenance processes. Hardware replacements, repairs, and board-level servicing can change a device's security state even when no malicious activity occurs. Replacing a processor or memory component may affect device identity, secure-boot provisioning, or calibration. An authorized repair procedure should specify which security properties must be revalidated before the aircraft returns to service. Where relevant, the assessment should verify that maintenance records identify the work performed and that the device's approved configuration is restored.

The quality of the final report depends on precise evidence and careful risk interpretation. Each finding should describe the affected component, the observed condition, the expected security requirement, and the practical consequence if the condition is exploited or causes a failure. Recommendations should be specific enough for engineering teams to act on, while respecting hardware limitations and operational requirements. A recommendation to disable every debug interface, for example, may be inappropriate if authorized recovery depends on a controlled diagnostic path. A more suitable approach may involve restricting access, requiring authentication, documenting maintenance use, and verifying the production configuration.

Findings should also distinguish security weaknesses from design choices that are legitimate under the system's threat model. An accessible test point is not necessarily a vulnerability, and a device that lacks a dedicated secure element is not automatically insecure. The relevant question is whether the implemented controls adequately protect the system's assets against credible threats. This requires considering the value of the information, the physical access available to potential attackers, the capabilities of the hardware, and the consequences of compromise.

Assessment records should be stored securely because they may contain sensitive board layouts, interface locations, configuration details, and information about protective controls. Reports should avoid publishing unnecessary operational details that could assist unauthorized access. Distribution should be limited to personnel who need the findings to make engineering, maintenance, or risk-management decisions. Where remediation is required, the organization should establish an owner, a target date, and a method for verifying that the correction addresses the original issue.

Reassessment is necessary after significant changes. A firmware update may alter console behavior or debugging restrictions, while a board revision may introduce new test points or change the storage architecture. A replacement component may support different security features from the original part. Changes to manufacturing or maintenance procedures may also affect how credentials are provisioned and how security settings are verified. The assessment should therefore be treated as part of the drone's lifecycle assurance process rather than a one-time inspection.

A well-executed non-destructive hardware assessment provides a documented understanding of how physical access interacts with the drone's security architecture. It identifies exposed interfaces, evaluates the protection of sensitive storage and device identity, examines the implementation of hardware security features, and highlights uncertainties that require further investigation. By using authorized procedures, preserving the integrity of the equipment, and grounding findings in observable evidence, engineers can improve hardware security without introducing unnecessary damage or compromising the safe operation of the aircraft.

# Authentication, Cryptography, and Access Control

## Device Identity, Operator Verification, and Mutual Authentication

Authentication is a fundamental requirement for securing unmanned aircraft because every trusted interaction depends on establishing who or what is requesting access. A drone may communicate with a ground control station, exchange telemetry with an onboard computer, receive instructions from an operator, or transmit information to a remote service. Each connection creates a question that must be answered before sensitive information is disclosed or a privileged operation is permitted: is the communicating party genuinely the device, user, or service it claims to be?

This question becomes particularly important when aircraft operate beyond the immediate supervision of their owners, communicate through several networks, or depend on automated services to execute missions. A system that accepts commands solely because they arrive through an expected communication channel may mistake an unauthorized party for a legitimate operator. Similarly, a ground station that trusts any aircraft presenting a familiar identification number may accept information from an impostor. Reliable authentication establishes a stronger basis for trust by verifying evidence associated with an identity rather than relying on appearance, network location, or an easily copied identifier.

Device identity and human identity serve different purposes. Device identity establishes which aircraft, computer, communication module, or service is participating in an exchange. Human identity establishes which person is attempting to use the system. These identities may be connected through an organizational account or an assigned equipment record, but they should not be treated as interchangeable. An authenticated drone does not prove that the person issuing commands is authorized to operate it, and an authenticated operator does not prove that the aircraft receiving those commands is the intended vehicle.

A practical security architecture therefore establishes identities at several levels. An aircraft may possess a unique device credential, while its flight controller and companion computer have separate identities or protected credentials appropriate to their responsibilities. An operator may authenticate through an individual account, and a maintenance technician may receive a different set of permissions. A remote management service may have its own service identity. These distinctions make it possible to determine which entity participated in an exchange and which authority it was permitted to exercise.

An identifier alone is insufficient for authentication. A serial number printed on an enclosure, a device name stored in software, or a network address can help identify equipment operationally, but such values may be copied or altered. Authentication requires evidence that an entity controls a credential associated with the claimed identity. This evidence might take the form of a cryptographic signature, a response generated using a protected secret, or another approved authentication mechanism. The verification process must also establish that the credential is trusted, current, and permitted for the requested purpose.

Operator verification requires similar care. Passwords remain common because they are straightforward to deploy, but their

security depends on password quality, storage, protection against automated guessing, and the handling of compromised accounts. A password entered into an imitation login interface may be disclosed even when the legitimate system uses strong encryption. For privileged drone operations, organizations should consider stronger authentication methods, especially when the operator can alter mission parameters, change security settings, access sensitive recordings, or control aircraft behavior.

Multifactor authentication can strengthen operator verification by requiring evidence from more than one category, such as knowledge, possession, or a qualifying biometric characteristic. Its value depends on the independence and security of the factors. Two passwords do not provide the same protection as two genuinely different factors, and an additional factor offers limited assurance if the surrounding process allows attackers to bypass it through account recovery or an unsecured maintenance interface. Organizations should select methods appropriate to their operational environment and ensure that emergency access procedures do not quietly eliminate the protection they intend to provide.

Mutual authentication is especially important when two systems must establish trust in one another. In a one-sided arrangement, a ground station may verify the aircraft while the aircraft accepts commands without confirming the identity of the ground station. This leaves an imbalance: the operator knows which device is responding, but the device may not know who is issuing instructions. Mutual authentication requires each party to verify the other according to an agreed security policy before establishing a trusted session or performing sensitive operations.

For example, a ground station and an aircraft may each possess a cryptographic credential issued or approved by a trusted provisioning authority. During connection establishment, each side presents evidence that it controls its credential. The other side validates that evidence and checks whether the associated identity is

authorized. Depending on the protocol, the exchange may also establish fresh session keys that protect subsequent communication. The details vary by system, but the essential principle remains that neither party should infer legitimacy merely from the fact that a connection has been established.

Authentication must also be resistant to replay. A replay attack occurs when an unauthorized party records a previously valid authentication message or command and attempts to use it again. If the receiving system accepts repeated messages without checking their freshness or context, an old instruction could be mistaken for a current request. Nonces, timestamps, sequence numbers, and session-specific cryptographic values can help address replay, provided they are implemented correctly and supported by appropriate synchronization and state-management procedures.

The design must account for interrupted communication. A drone may temporarily lose connectivity, switch networks, restart a subsystem, or reconnect after a period of degraded service. Authentication should remain meaningful across these transitions. Systems should not automatically treat every reconnecting device as trusted simply because it was trusted earlier. They should also avoid forcing unsafe operational behavior merely because a noncritical authentication service becomes unavailable. The correct response depends on the operation, the aircraft's autonomy, and the safety requirements established for the mission.

Authentication is not a substitute for safety engineering. A correctly authenticated operator may issue an inappropriate command, and a compromised but valid account may be used to perform unauthorized actions. Security controls must therefore combine identity verification with authorization, command validation, operational limits, and independent safety mechanisms. Authentication answers whether an entity has established its claimed identity; it does not, by itself, determine whether every requested action is safe or permitted.

# Cryptographic Keys, Certificates, and Secure Device Provisioning

Cryptography provides the mechanisms used to protect information, verify identities, and establish trust between systems that communicate over potentially untrusted channels. In drone security, its role extends beyond encrypting radio traffic. Cryptographic techniques can verify firmware authenticity, protect stored credentials, establish device identity, detect unauthorized message modification, and create evidence that a command originated from an entity controlling an approved credential.

Different cryptographic functions address different security requirements. Encryption protects confidentiality by making information unintelligible to parties without the required key. Message authentication provides evidence that a message was generated by an entity possessing an appropriate secret or key. Digital signatures support verification of origin and integrity using a public-key system. Cryptographic hashes help identify changes to data, although a hash alone does not prove authenticity when an attacker can replace both the data and its unprotected hash value.

Symmetric cryptography uses a shared secret key for operations such as encryption and message authentication. It can be efficient for protecting the continuous stream of telemetry and control messages exchanged during a flight. Its principal management challenge is that the participating parties must obtain and protect the same secret. If a fleet uses one shared key for every aircraft, compromise of a single device may expose the security of many others. Individual or carefully scoped keys reduce this concentration of risk.

Public-key cryptography uses a related pair of keys with different roles. A private key must remain under the control of its authorized owner, while a corresponding public key can be distributed to systems that need to verify signatures or perform other supported cryptographic operations. This arrangement can simplify identity verification across larger systems because a verifier does not necessarily need access to a shared secret held by every device it trusts. Nevertheless, public-key systems still require trustworthy methods for distributing, validating, and revoking public credentials.

A digital certificate binds an identity or set of attributes to a public key through a signed record issued by a trusted authority. In a properly designed system, a verifier checks the certificate's signature, validity period, intended usage, and relationship to an accepted trust anchor. It must also determine whether the certificate has been revoked or otherwise rendered untrustworthy, according to the system's policy. Merely receiving a certificate does not establish that the associated device is authorized to perform every operation.

Certificate management is particularly important for drone fleets. An organization may operate many aircraft, multiple ground stations, maintenance computers, and remote management services. Each component may require a different identity and level of authority. A well-designed certificate policy defines which entities may issue credentials, which identities they may certify, how long credentials remain valid, and which operations each credential is intended to support. It should also specify how compromised credentials are revoked and how legitimate equipment receives replacement credentials.

Secure device provisioning establishes identity and security settings before a device enters operational service. This process may occur during manufacturing, organizational acceptance, or controlled commissioning. It establishes the device's approved identity, configures relevant trust anchors, and ensures that necessary

cryptographic material is generated or installed through an authorized procedure. Poor provisioning can undermine otherwise strong cryptographic design because credentials may be exposed before the device begins operating.

Where practical, private keys should be generated within the device's protected hardware or a controlled provisioning environment that prevents unnecessary exposure. If keys must be transferred, the transfer should use an approved secure process with access restricted to authorized personnel and systems. Private keys should not be embedded in ordinary firmware images, copied into source repositories, or reused indiscriminately across unrelated devices. Provisioning records should provide sufficient traceability to establish which device received which credential without unnecessarily exposing the credential itself.

A device identity should be linked to a defined lifecycle. During commissioning, the organization confirms that the hardware and credentials correspond to an approved asset. During operation, the device authenticates to permitted services and uses only the privileges associated with its role. During maintenance, authorized procedures may renew credentials or update trust settings. At retirement, the organization removes the device from trusted inventories, revokes its credentials where appropriate, and ensures that sensitive data is handled according to retention requirements.

Cryptographic protocol design must also consider how session keys are established. Long-term identity keys are often used to authenticate an exchange rather than directly encrypt every message sent throughout the device's lifetime. An authenticated key-establishment protocol can create fresh session keys for a particular connection. Properly designed protocols may provide forward secrecy, meaning that compromise of a long-term private key does not automatically reveal previously recorded session traffic, subject to the protocol's assumptions and implementation. This property can

be valuable when drones transmit sensitive imagery, operational records, or mission details.

Key separation reduces the consequences of compromise. A key used to authenticate firmware should not automatically be accepted as a key for operator login or encrypted telemetry. Each key should have a defined purpose, and the systems using it should enforce that purpose. Separation can be implemented through distinct keys, separate cryptographic contexts, hardware-backed access policies, and protocol designs that prevent a credential valid in one context from being reused in another.

Algorithm selection matters, but strong algorithms cannot compensate for poor implementation. Cryptographic systems should use established, appropriately reviewed methods and reliable implementations suited to the device's computational capabilities. Engineers must account for secure random-number generation, correct certificate validation, key lifetimes, error handling, and the possibility that secrets may leak through logs, diagnostic interfaces, memory dumps, or poorly protected backups. Custom cryptographic algorithms should generally be avoided because their security is difficult to establish through informal review.

Secure provisioning should be tested as a complete process rather than assessed only by inspecting the final configuration. The organization should be able to establish that each device receives the intended identity, that unauthorized parties cannot freely enroll new devices, and that production credentials are not inadvertently shared with development systems. It should also be possible to identify and quarantine equipment whose provisioning status cannot be verified. These controls are especially important when hardware passes through several manufacturing, distribution, and maintenance organizations before reaching its operator.

# Authorization Boundaries Between Aircraft, Applications, and Operators

Authentication establishes identity, while authorization determines what an authenticated entity is permitted to do. This distinction is essential in drone systems because different users and software components require different levels of authority. An operator may be permitted to initiate an approved mission but not alter the aircraft's trust configuration. A maintenance technician may need diagnostic access without permission to retrieve sensitive customer recordings. A companion computer may process camera data without possessing unrestricted authority over flight-critical functions.

An effective authorization model begins by identifying the system's protected assets and the actions that can affect them. These may include flight-control commands, mission plans, navigation settings, firmware updates, payload data, communication credentials, maintenance functions, and safety-related configuration. The organization then defines which identities may perform which actions, under what conditions, and with what limitations. These decisions should reflect operational responsibilities rather than convenience alone.

Role-based access control is one approach. It assigns permissions to defined roles, such as pilot, fleet administrator, maintenance technician, mission planner, or auditor. Individual users receive the roles needed for their duties, while the role definitions establish which functions those users can access. This arrangement simplifies management when an organization has many users and aircraft. It also supports consistent review because permissions can be evaluated against documented responsibilities.

Role-based control is not sufficient for every situation. A user may be permitted to operate one aircraft but not another, or may be authorized to access a particular mission only during an assigned period. Attribute-based authorization can account for additional context, such as device identity, mission status, operating location, time, data sensitivity, or the current state of an approval process. These conditions should be chosen carefully so that legitimate operations remain understandable and predictable.

The principle of least privilege requires each user, application, and service to receive only the authority necessary for its intended function. This principle reduces the impact of a compromised account or software component. If a payload-processing application needs to analyze images, it should not automatically receive permission to modify flight-control parameters. If a logging service needs to record telemetry, it should not require the ability to change operator credentials. Permissions should be narrow enough that a defect in one component does not grant control over unrelated functions.

Separation of duties provides another layer of protection for high-impact operations. Certain changes may require approval from more than one authorized person or process. For example, enrolling a new privileged administrator, modifying fleet-wide trust settings, or approving a critical firmware release may warrant additional review. The exact requirements depend on the organization's size and risk profile, but important security decisions should not always depend on a single account with unrestricted authority.

The boundary between the aircraft and its companion computer deserves particular attention. The companion computer may run sophisticated applications for imaging, mapping, data analysis, or mission coordination. These applications can increase the aircraft's capabilities, but they also introduce software dependencies and services that may not require the same trust level as the flight

controller. If every application can issue unrestricted flight commands, a vulnerability in a non-flight-critical program may affect the aircraft's ability to operate safely.

A safer architecture places a clearly defined interface between companion computing and flight-critical control. The interface accepts only approved command types and validates the parameters required for each operation. Commands that affect critical behavior may be subject to additional checks, including limits on speed, altitude, geographic boundaries, operating mode, or mission state where appropriate. The flight controller should retain responsibility for enforcing the safety constraints assigned to it rather than assuming that upstream applications will always behave correctly.

Authorization boundaries must also extend to communication services. A service that transmits telemetry should not necessarily be permitted to issue commands. A remote monitoring platform may need read access to system status but no authority to change flight behavior. Similarly, a maintenance interface may be allowed to retrieve diagnostic records while restricting firmware installation to a separate, explicitly authorized procedure. Separating these functions limits the consequences of compromised credentials or vulnerable services.

Command authorization should be evaluated at the point where the action is executed, not solely at the user interface. A ground control application may hide controls that a user is not permitted to use, but hiding a control does not prevent another client from attempting the same request. The receiving system must enforce its own authorization policy. This principle applies to network APIs, onboard services, maintenance commands, and interfaces between embedded components.

A command should also be checked for context and validity. Even an authorized operator should not be able to submit arbitrary values that place the aircraft outside its approved operating limits. The

receiving system should verify the command's format, range, current applicability, and relationship to the aircraft's state. Authentication and authorization establish who may request an action; validation establishes whether the requested action is meaningful and acceptable under the current conditions.

Authorization decisions should be consistent across the system. If an operator loses a role or an account is suspended, the change should propagate to the services that rely on that identity within a defined period. Long-lived sessions and cached permissions can create a gap between administrative intent and actual enforcement. Systems should define how existing sessions are handled after access is revoked and which operations require renewed authorization.

Emergency procedures need special treatment. An aircraft may lose its connection to an authorization service, or an operator may need to respond to an urgent safety event. The system should have documented behavior for these situations rather than improvising permissions when connectivity fails. Some operations may continue under previously established, limited authority, while other actions may require immediate local authorization or be restricted until a trusted connection is restored. The correct choice depends on the aircraft's mission, autonomy, and safety requirements.

Good authorization design is supported by clear records. Sensitive actions should generate audit events identifying the authenticated actor, the operation requested, the relevant device, the decision made, and the result. Logs should avoid storing passwords, private keys, session secrets, or unnecessary personal information. Their integrity and availability should be protected so that they remain useful during incident investigation, compliance review, and maintenance.

# Credential Storage, Key Rotation, Revocation, and Recovery

Credential security extends beyond initial authentication. Drone systems operate over extended periods, may undergo maintenance, and can be transferred between operators or organizations. Credentials can expire, become compromised, or remain in circulation after a device has been retired. A reliable security program must therefore manage credentials throughout their lifecycle, including storage, renewal, revocation, recovery, and eventual destruction.

Credential storage should reflect the sensitivity of each item. Passwords used by a service should generally be stored as appropriately salted password hashes rather than as readable text. Cryptographic private keys require different treatment because they must remain usable for signing or authentication operations. Where supported, hardware-backed storage can reduce the opportunity for ordinary application software to read the key directly. Other secrets may be protected through operating-system access controls, encrypted storage, and tightly restricted service permissions.

No storage mechanism is effective if the surrounding application exposes the credential through an unnecessary interface. Secrets may appear in diagnostic output, configuration backups, crash reports, temporary files, or source-control history. Engineers should review the full path through which credentials are created, used, transmitted, logged, backed up, and removed. Protecting the primary copy while leaving secondary copies exposed provides incomplete security.

Key rotation replaces a credential with a new one according to a planned process. Rotation can limit the duration for which a

compromised or outdated key remains useful and can support changes in cryptographic policy. However, frequent rotation is not automatically beneficial if the process is unreliable or creates operational interruptions. The appropriate schedule depends on the key's purpose, exposure, lifetime, and the consequences of compromise.

Long-term identity keys, short-lived session keys, firmware-signing keys, and storage-encryption keys should not all be managed identically. Session keys may be refreshed automatically as part of a secure communication protocol. Device identity credentials may require renewal through a controlled enrollment process. Firmware-signing keys may be protected under especially restrictive procedures because compromise could affect every device that trusts the corresponding signing authority. Storage-encryption keys may require additional planning to ensure that protected information remains recoverable when authorized personnel need it.

Rotation procedures should support a controlled transition. In some systems, the new credential must be installed and verified before the old credential is removed. During that interval, the system must prevent unauthorized parties from exploiting overlapping trust relationships. The process should also account for aircraft that are offline when a rotation begins, devices in remote locations, and equipment that cannot be serviced immediately. These cases require defined policies for temporary status, reconnection, and eventual compliance.

Revocation is necessary when a credential is compromised, a device is lost, an operator leaves an organization, or a system no longer meets the requirements for trusted participation. Revocation communicates that a credential should no longer be accepted. The practical effectiveness of revocation depends on how relying systems obtain and enforce current status information. If a remote service continues accepting a revoked credential because it has not updated

its trust information, the administrative decision has not been fully enforced.

Offline aircraft create a particular challenge. A drone may remain disconnected from the network for an extended period and therefore be unable to retrieve the latest revocation information. The security design should specify what the aircraft can do while offline, which operations require current trust information, and how revocation status is checked when connectivity returns. It should not assume that every device can receive changes immediately. Where the consequences of accepting a compromised credential are significant, the organization may need operational procedures to isolate or suspend affected equipment until its status can be verified.

Credential recovery is another sensitive process. Users forget passwords, lose authentication devices, and occasionally lose access to their normal accounts. Hardware can fail, and cryptographic components may require replacement. Recovery mechanisms are necessary for service continuity, but they can become a weak point if they permit access without evidence comparable to the original authentication process.

An account recovery process should establish the requester's identity through approved evidence, record the recovery action, and limit the privileges available until verification is complete. High-impact accounts may require additional review or independent approval. Recovery should not rely solely on information that is easily obtained from public sources or already exposed through the user's compromised account. Support personnel should follow consistent procedures rather than granting access based on informal requests.

Device recovery requires different considerations. If a drone's credentials are lost or its secure hardware fails, the organization must determine whether the device can be re-provisioned safely. The procedure may involve verifying the asset's identity, confirming ownership, replacing a component, installing approved firmware, or

revoking the previous credentials before issuing new ones. The design should make it difficult for an unauthorized person with physical possession of the aircraft to use a recovery procedure to obtain a trusted identity.

Backups must be governed by the same security principles as primary credentials. A backup containing private keys or recovery secrets can become an attractive target if it is less protected than the operational system. Organizations should restrict backup access, protect stored copies, document restoration procedures, and periodically verify that recovery works as intended. A backup that has never been tested may fail when it is most needed, while a backup that contains obsolete credentials may reintroduce a security weakness.

Key compromise should trigger a structured response. The organization needs to identify which systems trust the affected key, which data or operations may have been exposed, and whether other credentials share the same trust relationship. A compromised device-specific key may require revoking one aircraft's identity, whereas a compromised signing key may require a broader response involving software validation, trust-anchor updates, and a carefully controlled release process. The scope of remediation should be based on the key's actual role rather than on a generic assumption that all keys have equivalent impact.

Credential destruction at the end of service life is equally important. Decommissioned equipment may retain private keys, network credentials, cached authentication tokens, or sensitive configuration. Returning a device to a manufacturer, selling it, or assigning it to another organization should not leave the previous owner's identity and permissions intact. A documented decommissioning process should remove or render inaccessible the credentials and information that are no longer needed, while preserving records required for legitimate operational or legal purposes.

Lifecycle management should be supported by an inventory of devices, credentials, owners, and relevant trust relationships. The inventory need not expose secret values; it should provide enough information to establish which credential belongs to which asset, when it expires, who is responsible for it, and how it can be revoked. Without this visibility, organizations may be unable to determine which aircraft are affected by a compromised key or whether an old credential remains trusted somewhere in the fleet.

## Evaluating Defenses Against Impersonation and Unauthorized Access

Evaluating authentication and access-control defenses requires more than confirming that a login screen exists or that communications are encrypted. The objective is to determine whether the system reliably distinguishes legitimate entities from impostors, limits each authenticated entity to its approved privileges, and responds appropriately when credentials become invalid or communication conditions change. Testing should reflect the system's actual architecture and be performed only on equipment, accounts, and networks for which the assessor has explicit authorization.

The assessment begins with a threat model. Engineers identify the entities that must be trusted, the assets requiring protection, and the ways an unauthorized party might attempt to gain access. Relevant threats may include stolen operator credentials, copied device identifiers, compromised service accounts, exposed private keys, replayed authentication messages, improperly configured maintenance interfaces, or excessive privileges assigned to applications. The assessment should prioritize threats according to their likelihood and potential impact rather than treating every theoretical weakness as equally important.

An identity inventory provides the basis for this work. The assessor should establish which identities exist for aircraft, flight controllers, companion computers, ground stations, operators, maintenance personnel, and remote services. Each identity should have a defined purpose and an associated authentication method. If a device can access a sensitive service without presenting verifiable identity evidence, the organization should determine whether that behavior is intentional and whether compensating controls are adequate.

Authentication testing should verify both successful and unsuccessful cases. Approved test accounts and test devices can be used to confirm that legitimate identities are accepted under the intended conditions and that unrecognized identities are rejected. The assessment should also examine whether a device with a valid identity but an expired, revoked, or incorrectly scoped credential is denied the operations that require current authorization. A system that accepts legitimate credentials but fails to enforce revocation does not provide complete lifecycle protection.

Mutual authentication deserves separate verification. The assessor should establish whether the aircraft verifies the identity of the ground station or remote service before accepting privileged commands, and whether the ground station verifies the aircraft before trusting its telemetry or granting access to sensitive functions. Tests should use controlled, authorized configurations to determine whether either side accepts an untrusted identity. The findings should distinguish between failure to authenticate, incorrect certificate validation, and a policy decision that deliberately permits limited unauthenticated information.

Certificate validation should be assessed for more than the presence of a certificate. The system must check that the certificate chains to an approved trust anchor, is valid for the intended use, and satisfies the relevant validity and revocation policy. It should also handle malformed, expired, or otherwise unacceptable credentials safely. If

certificate status cannot be checked while the aircraft is offline, the design should document the behavior and define which operations remain permissible under that limitation.

Replay resistance should be evaluated through controlled tests that do not affect operational flight. The assessor can examine protocol documentation, sequence-number handling, freshness checks, and session-establishment behavior, then validate them in a simulation or isolated test environment. The important question is whether an old authentication exchange or command can be accepted as a new authorized action. A protocol may use timestamps or sequence numbers, but those features are effective only if the receiver checks them consistently and handles resets, reconnections, and synchronization failures correctly.

Access-control evaluation should examine permissions at the enforcement point. A user interface may hide administrative functions from ordinary operators, but the underlying service must also reject unauthorized requests. In an approved test environment, the assessor should verify that an account with limited permissions cannot perform privileged actions merely by using a different supported client or submitting a request outside the normal interface. The test should remain within the agreed scope and avoid making changes to live aircraft or production services.

The same principle applies to software components. If a companion application is intended only to read telemetry or process payload data, the assessor should verify that its permissions do not unintentionally allow firmware installation, credential extraction, or unrestricted flight-control commands. Where the architecture uses separate services or communication channels, the assessment should determine whether those boundaries are enforced by the receiving component rather than relying on the cooperation of the requesting application.

Credential lifecycle tests should examine rotation, revocation, and recovery. An approved test credential can be rotated to confirm that the replacement becomes active and that the previous credential is rejected according to policy. A test account can be revoked to establish how quickly the change reaches relevant services and whether existing sessions remain active. Recovery procedures should be examined to determine whether they require appropriate verification and whether they create a route to privileged access that bypasses normal controls.

Rate limiting and account-lockout policies may reduce automated attempts to guess credentials, but their design must consider availability and recovery. An overly aggressive lockout can allow an attacker to prevent a legitimate operator from accessing the system simply by causing repeated failures. A poorly designed limit may be easy to bypass across multiple interfaces or accounts. The assessment should determine whether controls are proportionate, consistently enforced, and supported by monitoring that helps distinguish suspicious activity from ordinary mistakes.

Logging and alerting provide evidence of how the system responds to authentication failures and unauthorized requests. Useful records may include the identity presented, the affected service, the type of operation attempted, the decision made, and the time of the event. Logs should be protected against unauthorized modification and should avoid recording passwords, private keys, or other secrets. Where possible, security events should be correlated across the aircraft, ground station, and remote services so investigators can understand the sequence of events.

Testing must also consider operational resilience. Authentication services can become unavailable because of network interruptions, server failures, expired credentials, or configuration mistakes. The aircraft should have defined behavior for these conditions. Depending on the mission and safety requirements, it may continue

an already approved autonomous task, restrict new privileged commands, return to a safe operating condition, or require a trusted connection before proceeding. The chosen behavior should be documented and validated through controlled simulation or approved ground testing.

A security control should not be judged solely by whether it blocks unauthorized access. It must also preserve the availability and safe operation of the aircraft under legitimate conditions. For example, revocation may need to prevent a compromised operator account from issuing new commands while avoiding an abrupt response that would endanger an aircraft already in flight. This does not mean that security restrictions should be ignored during an emergency; it means that the system's response must be designed in advance and validated against realistic operating scenarios.

Findings should be recorded with sufficient evidence to support remediation. A useful report explains the affected component, the security requirement, the test conditions, the observed result, and the practical risk. It should avoid claiming that an entire system is insecure because one interface behaves unexpectedly, or that a system is secure merely because a limited set of tests succeeded. The report should identify the scope of testing and any areas that could not be verified.

Remediation should address the underlying design weakness rather than only the immediate symptom. If a device accepts an untrusted certificate, correcting the trust-validation policy may be more appropriate than adding an unrelated network filter. If an application possesses excessive privileges, narrowing its permissions and strengthening the boundary at the receiving service may provide a more durable solution. If credentials are shared across a fleet, introducing device-specific identity and a controlled provisioning process may reduce the impact of future compromise.

After changes are implemented, the relevant tests should be repeated to confirm that the intended protections work and that legitimate operations remain functional. Regression testing should examine related interfaces because authentication and authorization changes can affect maintenance, telemetry, autonomous operation, and recovery procedures. The organization should also update its security documentation, credential inventory, and maintenance instructions so that the corrected design remains effective throughout the aircraft's service life.

# Autonomous Missions and Software Supply Chain Integrity

## Mission Planning, Waypoint Execution, and Geofence Configuration

Autonomous flight has transformed the way unmanned aircraft perform inspection, surveying, agricultural monitoring, environmental observation, infrastructure assessment, and other complex operations. Instead of relying on continuous manual control, an aircraft can follow a predefined route, collect information at designated locations, adjust its behavior according to sensor inputs, and complete an approved task with limited operator intervention. These capabilities depend on a combination of mission-planning software, flight-control logic, navigation systems, onboard configuration, and communication services. Each component contributes to the aircraft's behavior, which means that the security of an autonomous mission cannot be separated from the integrity of the information and software that govern it.

A mission plan is more than a collection of geographic coordinates. It can define the sequence of waypoints, target altitudes, speed limits, camera actions, waiting periods, changes in operating mode, and conditions under which the aircraft should continue or terminate an activity. Depending on the platform, it may also include parameters governing takeoff, landing, return behavior, payload operation, and responses to communication loss. The exact structure varies between

flight-control systems, but the principle is consistent: mission data influences how the aircraft behaves, sometimes without a human reviewing every action as it occurs.

This creates a security concern that differs from the traditional protection of communication channels. An encrypted connection can protect a mission file while it is transmitted, yet the aircraft may still execute an unsafe mission if the file was modified before encryption or generated by an unauthorized application. Similarly, an authenticated operator may unintentionally upload a plan containing incorrect coordinates, inappropriate altitude limits, or commands that conflict with the aircraft's operating requirements. Secure autonomous operation therefore requires controls over both the identity of the person or software supplying a mission and the validity of the instructions being supplied.

Mission planning should begin with clearly defined operational requirements. The intended route, survey area, altitude constraints, payload actions, and recovery conditions should be established before the mission is transferred to the aircraft. These requirements should reflect the aircraft's capabilities, applicable operating restrictions, environmental conditions, and the consequences of a navigation or control failure. A route suitable for an open agricultural field may be inappropriate for a dense urban area or a site containing power lines, cranes, or other obstacles. Security validation must preserve these operational distinctions rather than treating every technically valid route as an acceptable one.

Waypoint execution depends on the relationship between the planned route and the aircraft's navigation estimate. A waypoint may specify a horizontal position and an altitude, while associated commands determine what the aircraft should do when it reaches that location. The flight controller interprets the mission and converts its requirements into control actions. Errors in the mission representation, unexpected changes to configuration, or incorrect

assumptions about coordinate systems can therefore affect the aircraft's actual behavior.

Coordinate handling deserves particular attention. Mission-planning applications and flight controllers must agree on the coordinate reference system, altitude reference, units, and interpretation of waypoint parameters. A horizontal coordinate may be expressed using geographic latitude and longitude, while altitude may be relative to the takeoff point, referenced to a specified datum, or derived from another supported reference. Confusing these conventions can produce a route that appears reasonable in one application but leads to unintended behavior when executed by another component. Mission validation should verify that the interpretation used by the receiving aircraft matches the assumptions made during planning.

Waypoint sequencing is equally important. A mission can contain valid individual waypoints but still be unsuitable as a complete route because of abrupt changes in direction, excessive travel distance, conflicting actions, or inadequate clearance from obstacles. A mission validator should examine relationships between successive points and assess whether the requested sequence fits the aircraft's performance limits. Where terrain, structures, or dynamic obstacles are relevant, a route should not be considered safe merely because its coordinates fall within a permitted geographic area.

Geofencing provides an additional layer of operational control by defining geographic boundaries within which an aircraft is permitted or expected to operate. A geofence may describe a permitted area, a restricted region, an altitude ceiling, or a combination of horizontal and vertical limits. Depending on the system, crossing a boundary may generate a warning, prevent a mission from starting, trigger a navigation response, or activate another predefined safety procedure. These behaviors should be understood and tested rather than assumed from the mere presence of a geofence setting.

A geofence is only as reliable as the data, configuration, and enforcement mechanisms supporting it. If its boundaries are entered incorrectly, stored in an unprotected configuration file, or interpreted using inconsistent coordinate references, the aircraft may enforce the wrong limits. If the boundary is checked only by the mission-planning application, a different software component may be able to submit a route that bypasses the intended restriction. For meaningful protection, critical operating limits should be checked by the system responsible for enforcing them during flight, with suitable safeguards against unauthorized configuration changes.

Geofence behavior also requires careful treatment when navigation quality deteriorates. An aircraft cannot reliably determine whether it is inside a geographic boundary if its position estimate is substantially inaccurate or untrustworthy. A robust design should consider how navigation uncertainty affects boundary enforcement and what actions are appropriate when position integrity cannot be established. Depending on the aircraft and operating conditions, the response may involve restricting further autonomous movement, requesting operator intervention, or following a previously validated contingency procedure.

Mission authorization should account for more than the operator's general permission to use the aircraft. An operator may be authorized to conduct a survey in one location but not to alter the geofence, remove an altitude restriction, or load a mission intended for another aircraft. The system should distinguish between permission to create a mission, permission to approve it, and permission to execute it. For higher-consequence operations, a review process may be appropriate before mission data is released for use.

Changes made after approval must also be controlled. A route may be modified because of weather, a new obstacle, a revised survey requirement, or a temporary restriction. Those changes should not

silently invalidate the assumptions under which the original mission was approved. The organization should establish which modifications require renewed validation, who may authorize them, and how the aircraft confirms that it has received the current approved version.

Autonomous missions also depend on failure handling. The aircraft may encounter low battery capacity, unexpected wind, navigation uncertainty, loss of communication, sensor faults, or a discrepancy between the planned route and the actual flight path. Mission logic should define suitable responses within the aircraft's verified capabilities. Security measures must not prevent legitimate safety actions, and a cybersecurity event should not cause an uncontrolled response that creates a greater hazard than the original problem.

The most reliable approach treats mission planning as a controlled engineering activity. Routes are created from defined requirements, validated against technical and operational constraints, approved according to risk, and transferred through authenticated channels. The aircraft independently checks the received mission before execution and continues to enforce critical safety boundaries while flying. This layered approach reduces the likelihood that a planning mistake, compromised workstation, or unauthorized configuration change will translate directly into unsafe autonomous behavior.

## Validating Mission Files and Protecting Flight Configuration Integrity

Mission files and flight configuration records provide the information that autonomous systems use to determine how they should operate. Depending on the platform, these files may contain waypoint coordinates, mission commands, navigation parameters, geofence definitions, payload settings, failsafe behavior, and other

values that influence aircraft operation. Some information may be stored in a file uploaded before flight, while other parameters reside in persistent memory or are transferred through a ground control application. Because these values can affect safety and security, their integrity must be protected throughout creation, approval, transmission, storage, and execution.

A mission file should be treated as structured operational input rather than trusted merely because it was generated by a familiar application. Files may be malformed, incomplete, outdated, created for a different aircraft, or modified after approval. Some errors arise from ordinary human mistakes, while others may result from compromised software or unauthorized changes. A secure implementation should therefore validate mission content independently of the process that created it.

Structural validation establishes whether a file follows the expected format. The receiving software should confirm that required fields are present, values have the correct data types, and the file does not contain unsupported structures that could cause incorrect interpretation. It should reject malformed records, invalid numeric values, unreasonable file sizes, and command types that the aircraft does not support. This is important because parsers often process data from external applications, and an assumption that uploaded files are always well formed can expose the receiving system to errors or vulnerabilities.

Semantic validation goes beyond file structure. A mission may be syntactically valid while containing coordinates outside the approved operating area, an altitude inconsistent with the aircraft's limits, or commands that are incompatible with the current flight mode. The validator should evaluate whether each command is meaningful in context and whether the mission as a whole satisfies the applicable restrictions. It should also check that required contingency behavior is defined where the operational design requires it.

Validation should use authoritative constraints. If the aircraft has a documented maximum operating altitude, a mission should not be accepted merely because the planning application allows a higher value. If the approved operating area has defined boundaries, the receiving system should compare the mission against those boundaries using consistent coordinate and altitude conventions. Limits should be derived from controlled configuration or another trusted source, not from values that an untrusted mission file can redefine for itself.

Compatibility checks are also necessary. A mission created for one aircraft model or firmware version may contain commands that another system interprets differently or does not support. A receiving aircraft should establish that the mission is compatible with its software capabilities, payload configuration, and operational mode. Unsupported commands should not be silently ignored when doing so could change the mission's meaning. The system should reject the file or require an authorized correction, depending on the significance of the discrepancy.

Mission authenticity and mission validity are separate properties. A cryptographic signature can establish that a file was approved by a holder of a trusted signing key and that its signed contents have not changed. It cannot establish that the approved mission is safe, correctly designed, or appropriate for the current conditions. Conversely, a mission that satisfies every numerical constraint may still be unauthorized. Strong mission security combines authenticity verification with semantic validation and explicit authorization.

Where signatures are used, the signed data should cover all security-relevant mission content. If a mission consists of several files or references external configuration records, the verification process must establish which components belong to the approved package. Otherwise, an attacker or an accidental process may replace an unsigned supporting file while leaving the signed mission document

intact. The system should define precisely what is covered by the signature and reject ambiguous combinations that could cause different components to interpret the same mission differently.

Cryptographic hashes can help identify a particular mission version and detect changes when the expected hash is obtained through a trusted process. However, a hash value stored beside a file in an unprotected location is not sufficient to establish authenticity because both may be replaced. The trusted reference must come from a protected approval process, a verified signature, or another mechanism that prevents unauthorized substitution.

Version management prevents confusion between approved and obsolete mission files. A mission should have an identifiable version, an associated aircraft or operational context, and a record of its approval status. If a route is revised, the new version should be distinguishable from earlier copies. The aircraft or mission-management service should make it clear which version is scheduled for execution, reducing the risk that an operator inadvertently loads a file that was superseded after a safety review.

Flight configuration integrity extends beyond mission files. Persistent settings may define navigation behavior, communication parameters, geofence enforcement, sensor calibration, battery thresholds, and response to communication loss. Unauthorized changes to these values can alter how the aircraft behaves even if the mission itself remains unchanged. Configuration management should therefore establish which values are security-critical, which roles may modify them, and how approved settings are restored after maintenance or software updates.

Configuration baselines provide a useful reference. A baseline records the expected state of the aircraft for a defined model, hardware revision, firmware release, or operational role. The system can compare the current configuration against this approved state and identify unexpected changes. Not every difference indicates an

attack; some may result from legitimate maintenance or adaptation to local operating conditions. The process should distinguish authorized deviations from unexplained changes and require review where the difference affects safety or security.

The ability to modify configuration should be limited according to responsibility. An operator who can upload a mission may not need permission to change cryptographic trust settings, disable safety restrictions, or alter the aircraft's update policy. A maintenance technician may need access to calibration settings but not to sensitive mission records. These boundaries reduce the chance that a compromised account or an accidental selection will alter unrelated aspects of the system.

Audit records provide evidence of how mission and configuration data change over time. Useful records may identify who created or approved a mission, which version was uploaded, which configuration values were changed, and whether validation succeeded. The records should be protected against unauthorized modification and should avoid unnecessarily retaining sensitive information. Their integrity matters because they may be needed to investigate an unexpected flight path or establish whether a configuration change occurred before an incident.

Mission and configuration validation should also consider interrupted updates. If a transfer fails halfway through, the aircraft should not be left with a mixture of old and new settings that produce an undefined state. Where appropriate, updates should use a transactional process in which the new configuration is validated before becoming active, and the previous approved state remains available until the change is completed successfully. Recovery behavior must be designed carefully so that rollback does not restore a configuration known to be unsafe or obsolete.

These measures are most effective when validation occurs at multiple stages. The planning application can detect obvious mistakes, an

approval service can verify policy compliance, and the aircraft can independently validate the data it receives. Each stage serves a different purpose. Earlier checks reduce operator errors, centralized review provides governance, and onboard validation protects against errors or changes that occur after approval. The receiving aircraft should not rely solely on a remote application to enforce critical constraints.

# Third-Party Libraries, Dependencies, and Firmware Build Pipelines

Modern drone software rarely consists entirely of code written by the manufacturer or project team. Flight-control applications, ground stations, companion computers, communication services, and development tools may rely on external libraries, operating-system components, protocol implementations, device drivers, compiler toolchains, and automated build services. These dependencies reduce development effort and provide established functionality, but they also introduce security and maintenance obligations. A weakness in a component used indirectly by a drone application can affect the resulting system even when the application's own source code appears well designed.

A software dependency is any external component required to build, package, or run a program. Direct dependencies are explicitly included by the project, while transitive dependencies are brought in by those components. A single library may therefore introduce several additional packages that developers do not use directly or recognize by name. If the dependency structure is not recorded, an organization may struggle to determine which products are affected when a vulnerability is disclosed.

Dependency management should begin with visibility. Developers need to know which components are used, which versions are included, where they came from, and why they are required. A software bill of materials can support this task by recording relevant component names, versions, and relationships within a software product. Depending on the format and build process, it may also contain supplier information, package identifiers, or references to licensing and security records. Such documentation helps engineering and security teams assess exposure when a component is found to contain a vulnerability.

A bill of materials is not a guarantee that software is secure. It may be incomplete, inaccurate, or disconnected from the binary actually installed on the aircraft. Its value depends on the quality of the build records and the relationship between the inventory and the released artifact. Organizations should generate or verify component inventories as part of the release process and retain enough information to connect the inventory to a specific firmware version.

Dependency selection should account for maintenance and security history as well as functionality. A component may satisfy immediate performance requirements but be unsuitable if it is abandoned, poorly documented, difficult to update, or dependent on unsupported software. The choice should consider whether security fixes are available, whether the maintainers respond to reported vulnerabilities, and whether the project can replace the component if continued use becomes unacceptable.

Version pinning can improve repeatability by ensuring that builds use specified dependency versions rather than whatever happens to be current at build time. However, pinning an outdated component indefinitely can preserve known vulnerabilities. A responsible process combines controlled versions with regular review, vulnerability assessment, and planned updates. Changes should be

evaluated for compatibility and regression risk before deployment to aircraft.

Dependency sources also require verification. Public package repositories and source-code hosting services make software widely available, but the name of a package alone does not prove that it originated from the expected maintainer. A lookalike package, compromised maintainer account, malicious release, or altered download can introduce harmful code into the build. Organizations should use approved sources, verify cryptographic checksums or signatures when supported, and restrict the ability to introduce new dependencies into production software.

Firmware build pipelines deserve particular protection because they transform source code and dependencies into the executable artifacts installed on aircraft. A pipeline may include source retrieval, dependency resolution, compilation, linking, testing, signing, packaging, and release publication. If an attacker compromises any sufficiently privileged stage, the resulting firmware may be altered even when the original source repository remains unchanged.

Build systems should therefore be treated as sensitive infrastructure. Access to repositories, build workers, artifact stores, signing services, and release controls should be limited to authorized identities. Administrative privileges should be separated where practical, and changes to pipeline definitions should receive appropriate review. Automated jobs should operate with only the permissions needed for their tasks rather than using broad credentials that provide access to unrelated systems.

Build secrets require particular protection. A pipeline may need credentials to retrieve private dependencies, publish artifacts, or request a release signature. These secrets should be stored in an appropriate protected mechanism and made available only to the jobs that require them. They should not be embedded in source code, printed in logs, or included in downloadable build artifacts. A

compromise of the build environment should not automatically expose every credential used across the development organization.

Build isolation reduces the impact of untrusted code. Dependency installation scripts and build tools may execute code during compilation or packaging, depending on the ecosystem and configuration. Running these operations in controlled, disposable environments can limit their access to unrelated systems and sensitive data. Network access can also be restricted where feasible so that builds use approved inputs instead of retrieving uncontrolled resources during compilation.

Automated security checks can identify known vulnerabilities, suspicious dependency changes, insecure configurations, and certain classes of coding errors. Static analysis examines source code without executing the program, while dynamic analysis evaluates behavior during execution. Software composition analysis focuses on external components and their known issues. These techniques provide useful evidence, but none can prove the absence of vulnerabilities. Their results must be interpreted in context and combined with code review, testing, and operational risk assessment.

Vulnerability severity should be assessed against actual use. A library flaw may be serious in a network-facing service but irrelevant to a component that is not included in the released product or whose vulnerable function cannot be reached under the deployment configuration. Conversely, a vulnerability in a low-level parser or communication component may deserve urgent attention if it processes untrusted input in a safety-critical system. Prioritization should account for exploitability, exposure, available mitigations, and potential consequences for aircraft behavior.

Updates require a disciplined release process. A dependency should not be upgraded solely because a newer version exists, nor should it remain unchanged simply because the current firmware has passed previous tests. Developers should assess the change, verify the

source, rebuild the software, run relevant tests, and document the resulting artifact. For flight-critical components, regression testing should cover expected behavior under normal conditions as well as faults that could affect navigation, control, or recovery.

## Software Provenance, Signed Releases, and Reproducible Builds

Software provenance describes where a software artifact originated, how it was produced, and which inputs and processes contributed to its creation. For drone firmware, provenance can connect the installed executable to a particular source revision, dependency set, toolchain, build environment, test record, and release approval. This information becomes especially important when an aircraft behaves unexpectedly or when a vulnerability affects a component used across multiple products.

Without reliable provenance, an organization may know that a firmware file is installed but be unable to establish which source code produced it or whether it passed the expected release process. Two files bearing the same version label may have been built from different source revisions or with different dependencies. A build record should therefore identify artifacts through reliable identifiers, such as cryptographic hashes, and link them to the corresponding source and build information.

Provenance records should be generated through controlled processes rather than assembled entirely by hand after a release. Automated build systems can record the source revision, dependency versions, compiler and toolchain details, build configuration, and identity of the process that produced the artifact. Where appropriate, the record may also identify test results and approval events. The

purpose is to make the software's history traceable and to support independent verification.

Digital signatures help protect released firmware against unauthorized modification and substitution. A signing process uses a protected private key to create a signature over the artifact or a defined manifest of its contents. A receiving system verifies the signature using the corresponding trusted public key or another approved trust mechanism. If the signed content changes, verification should fail, provided the implementation correctly covers the relevant data and validates the signature.

The trustworthiness of a signature depends on more than the mathematical algorithm. The signing key must be protected, the signing process must be authorized, and the verifier must use the correct trust policy. If an attacker can obtain the private signing key or cause the release service to sign an unapproved artifact, a technically valid signature may authenticate software that should never have been deployed. Strong release security therefore combines cryptographic verification with controlled approval and protected signing infrastructure.

Signing keys should be separated from ordinary development credentials. Developers may need to submit changes or initiate test builds, but those permissions should not automatically authorize them to sign production firmware. A release process can require successful automated checks, appropriate review, and approval from authorized personnel before the signing operation becomes available. Higher-risk releases may warrant separation between the person proposing the change and the person or system authorizing publication.

Firmware verification should occur on the receiving device whenever the hardware supports it. If the aircraft relies solely on a ground station to verify a firmware file before transmission, an attacker who bypasses that workstation or substitutes the file afterward may

undermine the protection. Onboard verification provides an independent control by ensuring that the firmware accepted for installation satisfies the aircraft's own trust policy.

The verification policy should cover the relevant components of the update package. Some aircraft use a single firmware image, while others update multiple processors, boot components, radio modules, or companion-computer applications. If the release process signs only one part of the package, another component may remain insufficiently protected. A manifest or equivalent mechanism can define the complete approved release and establish the relationship between its components.

Rollback protection is also important. A correctly signed firmware release may still contain an old vulnerability that was fixed in a later version. If the aircraft accepts any historically signed image, an attacker may attempt to install an older release that restores a known weakness. A version policy can restrict installation to releases that satisfy the current security requirements. Implementing this policy requires care because legitimate recovery may sometimes require reinstalling a previous version. Such exceptions should be controlled and documented rather than left to unrestricted downgrade behavior.

Reproducible builds offer another way to improve confidence in software artifacts. A build is reproducible when the same declared source inputs, build instructions, dependencies, and relevant environment produce the same output under the defined reproducibility conditions. This allows an independent party to rebuild software and compare the result with the distributed artifact. A matching result provides evidence that the published binary corresponds to the stated inputs and build process.

Reproducibility is not always straightforward. Build outputs may contain timestamps, nondeterministic ordering, environment-specific paths, or generated information that changes between runs.

Compilers and packaging tools may also introduce differences that are unrelated to the program's functional behavior. Achieving reproducibility requires deliberate engineering, controlled dependencies, documented toolchains, and an understanding of which parts of the output are expected to remain identical.

A reproducible build does not prove that the source code is safe. If the source contains a vulnerability, independent builds can reproduce the same vulnerable result. Nor does reproducibility alone establish that the declared source repository is trustworthy. Its principal value is to make the relationship between source inputs and released binaries easier to verify, reducing uncertainty about whether an unexplained build modification occurred.

Independent verification can strengthen this process. A separate build environment can retrieve the approved source and dependencies, apply the documented build instructions, and compare the result against the released artifact. Any unexplained difference should be investigated before the release is treated as verified. The process may be especially useful for safety-critical software, widely deployed firmware, or systems where confidence in the build infrastructure is a significant concern.

Release records should remain available throughout the supported lifetime of the aircraft. When a security advisory identifies a vulnerable library or a firmware release must be withdrawn, provenance information helps determine which aircraft and software versions are affected. It also supports the creation of a corrected release by identifying the relevant source revision, build process, and dependency changes. Without these records, incident response may depend on incomplete recollections or uncertain version labels.

Software signing and provenance should be integrated with update authorization. The aircraft needs to establish not only that an image was signed by an accepted key, but also that the release is appropriate for that hardware, compatible with the required

configuration, and permitted under the current update policy. A signature should not automatically grant authority to install arbitrary software on every device that trusts the same signing infrastructure.

## Securing Autonomous Workflows Against Configuration and Supply Chain Risks

An autonomous drone workflow includes every stage through which an operational task moves from planning to execution and review. It may begin with a mission request, proceed through route design and configuration, pass through approval and software preparation, and end with mission upload, autonomous execution, and data collection. Different applications, operators, services, and hardware components may participate at each stage. Security depends on preserving trust as information moves between these participants and ensuring that no stage can silently introduce changes that invalidate earlier checks.

A secure workflow begins with defined responsibilities. The person who creates a mission may not be the person who approves it, and the software that generates a route should not automatically possess permission to change the aircraft's security configuration. Likewise, a developer who modifies firmware should not necessarily be able to release it directly to an operational fleet. Separating these responsibilities reduces the consequences of compromised accounts and provides an opportunity to identify mistakes before they reach the aircraft.

The workflow should maintain a reliable relationship between requirements, approved configuration, and deployed software. A mission may be designed for a specific aircraft model and firmware version, with assumptions about available sensors, navigation

capabilities, payload equipment, and geofence behavior. If the aircraft receives a different software release or configuration, those assumptions may no longer hold. Compatibility checks should therefore verify that the mission, firmware, and relevant operational settings form an approved combination.

Configuration drift is a recurring concern in systems that remain in service for extended periods. Drift occurs when the current state of a device gradually differs from its approved baseline because of maintenance, software updates, manual adjustments, or unrecorded changes. Some deviations are legitimate, but unexplained differences can weaken security controls or change flight behavior. Organizations should maintain an authoritative configuration record and periodically compare deployed systems against the expected state.

Automated deployment can reduce manual errors, but it must be designed with appropriate safeguards. A pipeline that distributes mission files or firmware across a fleet should verify the destination aircraft, check the approved release, and confirm that each device satisfies the relevant compatibility requirements. The pipeline should not rely solely on a device name or network address that can be copied. Strong identity verification and explicit authorization should govern the delivery process.

The same workflow must protect against unauthorized changes between approval and execution. A mission may be approved on one workstation, copied to removable storage, transferred through a network service, and uploaded from another computer. Each transfer introduces a possibility of accidental corruption or substitution. Cryptographic verification, version control, protected storage, and onboard validation help preserve the approved content throughout this sequence.

Software supply chain security must extend to development workstations and build infrastructure. A compromised developer

account, malicious dependency, or altered build configuration can introduce changes that are not obvious during routine operation. Organizations should restrict administrative access, review changes to security-critical files, monitor dependency updates, and protect release credentials. Build artifacts should be linked to verifiable source revisions and controlled release records before they are approved for operational deployment.

The integrity of update delivery is particularly important for autonomous aircraft. Updates may correct vulnerabilities, improve reliability, or add new functions, but an incorrectly applied update can also disrupt flight behavior. A secure update process should authenticate the package, verify its integrity, check compatibility, and apply it through a controlled procedure. It should also define what happens if power is interrupted, validation fails, or the aircraft cannot complete the update. Recovery must preserve security rather than creating a permanent unrestricted maintenance mode.

Operational separation can help protect production aircraft from development activity. Experimental firmware, new mission logic, and unverified dependencies should be evaluated on designated test equipment or in a simulation environment before they are approved for operational use. Testing should progress through stages appropriate to the risk, including software-level checks, hardware-in-the-loop evaluation where available, ground testing, and carefully controlled flight trials when necessary and authorized. A successful simulation is valuable evidence, but it does not establish that every physical condition has been covered.

Configuration and supply chain controls should account for third-party service providers. An organization may rely on external developers, maintenance companies, mapping services, data-processing platforms, or hardware suppliers. Each relationship introduces dependencies that should be documented and governed according to risk. Contracts and technical procedures can define how software is delivered, how vulnerabilities are reported, how

credentials are protected, and how the organization is notified when a component or service is compromised.

Incident response should be prepared for both mission-level and supply-chain events. An unexpected route change may result from an operator mistake, corrupted mission data, compromised planning software, or unauthorized access to an account. An unexplained firmware difference may result from an interrupted update, a manufacturing error, or a malicious modification. Investigation should preserve available evidence, compare the affected system with its approved baseline, and avoid assuming a cause before sufficient information is available.

When a component is found to be compromised, the response should identify the scope of affected devices and workflows. A vulnerable dependency may exist in several firmware releases, while a compromised signing credential may affect every product that trusts the corresponding key. The organization should use its software inventory, provenance records, device inventory, and release history to determine which systems require inspection or remediation. Where the extent of compromise is uncertain, affected components may need to be isolated from sensitive operations until their status can be established.

Recovery should restore a known, approved state. This may require replacing mission files, correcting geofence settings, reinstalling verified firmware, rotating credentials, or revoking trust in a compromised release key. Merely restarting the aircraft or re-uploading the last available mission may not resolve the underlying problem. The response should also verify that unauthorized changes have not persisted in configuration storage or secondary onboard components.

The organization should review the workflow after significant changes to hardware, software, suppliers, or operating requirements. A new payload may introduce additional dependencies, a firmware

update may change how mission commands are interpreted, and a revised maintenance process may alter who can access production equipment. Security assumptions should be revisited when the system changes rather than preserved indefinitely from the original design.

Continuous assurance depends on evidence. Mission approval records, configuration baselines, software bills of materials, signed release manifests, test results, and maintenance histories each provide a different view of system integrity. When these records are connected, the organization can trace a mission to its approved plan, the aircraft to its current configuration, and the installed firmware to its documented build process. This makes it easier to identify discrepancies, respond to newly discovered vulnerabilities, and establish whether an aircraft remains suitable for its intended operation.

A mature autonomous workflow treats security checks as part of normal engineering and operational practice. Mission data is validated before execution, critical restrictions are enforced onboard, software dependencies are tracked, releases are verified, and privileged changes are limited to authorized roles. These measures do not eliminate every possible failure, but they reduce opportunities for accidental or unauthorized changes to pass unnoticed from planning systems and software suppliers into the behavior of an autonomous aircraft.

# Vulnerability Research and Security Validation

## Defining Assessment Scope, Authorization, and Rules of Engagement

Vulnerability research is a disciplined process for identifying weaknesses in a system, determining the conditions under which those weaknesses matter, and establishing whether they can affect confidentiality, integrity, availability, or safe operation. In unmanned aircraft, this work requires a broader perspective than examining software for coding mistakes. A drone is a connected cyber-physical system in which software, electronics, radio communications, navigation sensors, operator interfaces, and physical mechanisms interact. A weakness in one component may remain relatively contained, or it may influence the behavior of several other components through their dependencies.

Security validation provides the evidence needed to determine whether protective measures work as intended. A design document may state that a communication channel requires authentication, for example, but that statement alone does not establish that every relevant service enforces authentication correctly. A firmware specification may describe secure startup checks, yet the implementation may contain configuration errors or fail to reject an unacceptable image. Validation examines the difference between intended protection and observed behavior, using controlled methods that produce findings engineers can reproduce and address.

The first responsibility in any assessment is to establish a precise scope. Scope identifies the aircraft, software, hardware, networks, services, interfaces, and configurations that the assessment is permitted to examine. It also distinguishes systems that belong to the assessment from those that merely communicate with them. This distinction is especially important when a drone depends on external infrastructure, including cloud services, cellular connectivity, mapping platforms, maintenance systems, or third-party software repositories. Permission to assess an aircraft does not automatically grant permission to test every external service on which it depends.

An assessment should begin by documenting the system under examination. Relevant information includes the aircraft model and hardware revision, flight-controller architecture, firmware versions, companion-computer software, communication interfaces, ground-control applications, mission-management services, and relevant security configurations. The level of detail should be sufficient to identify the components being evaluated and understand their relationships. Where some information is unavailable, the assessor should record that uncertainty rather than silently assume a particular implementation.

Authorization must be explicit and appropriate to the proposed activities. Ownership of a drone does not necessarily establish permission to assess a network operated by another organization, a supplier's cloud infrastructure, or a third-party application service. Likewise, permission to inspect a device does not automatically authorize destructive testing, permanent changes to hardware, extraction of sensitive data, or flight experiments that could endanger people or property. The assessment agreement should make these boundaries clear before testing begins.

Rules of engagement establish how the assessment will be conducted. They define the permitted techniques, testing windows, points of contact, operational restrictions, evidence-handling

requirements, and procedures for stopping work when unexpected conditions arise. For drone systems, these rules should account for physical safety as well as cybersecurity. Testing that interrupts a communication interface may be acceptable on a bench but inappropriate during a live mission. An experiment that affects navigation data should be performed in a controlled simulation or other approved environment rather than near an aircraft carrying out an operational task.

The rules should also identify which activities require additional approval. Examples include changing persistent configuration, modifying firmware, connecting to previously undocumented interfaces, testing authentication limits, or investigating a suspected security weakness that may affect flight-critical behavior. Establishing these boundaries in advance prevents an assessor from making improvised decisions under pressure and helps the system owner understand the possible consequences of each test.

A useful assessment distinguishes discovery from exploitation. Discovery seeks evidence that a potential weakness exists, while exploitation attempts to demonstrate that the weakness can produce a particular security consequence. In many cases, the first objective can be achieved through code review, configuration inspection, simulation, or non-invasive testing. More intrusive proof may be unnecessary if the available evidence already establishes a serious design flaw. The appropriate level of testing should be determined by the value of the evidence and the risk of causing damage.

The assessment environment is another important consideration. Production aircraft may contain sensitive operational data, depend on carefully calibrated sensors, or be difficult to restore after an unsuccessful test. A dedicated test unit, hardware-in-the-loop setup, software simulation, or isolated laboratory can provide a safer setting for investigating behavior. The selected environment should resemble the relevant production configuration closely enough for

the findings to be meaningful, while remaining separated from operational equipment and networks.

A laboratory configuration should be documented so that results can be interpreted correctly. Hardware revisions, firmware builds, configuration values, connected peripherals, and test tools may affect observed behavior. If a weakness appears only on one software release or depends on a particular setting, that condition is part of the finding. Without this context, engineers may be unable to reproduce the result or determine whether it applies to the deployed fleet.

Scope management must continue throughout the assessment. New evidence may reveal that a component is outside the original boundary or that a test could affect an external system. The assessor should pause the relevant activity, explain the finding to the responsible contact, and obtain appropriate authorization before expanding the work. A technically interesting discovery does not justify ignoring the agreed limits.

Confidentiality and evidence handling also form part of the engagement. Security assessments may expose credentials, proprietary designs, personal information, mission records, or operational details. The assessor should collect only the evidence needed to support the findings, store it securely, and limit access to authorized recipients. Reports should describe the weakness clearly without reproducing secrets or publishing unnecessary details that could increase risk.

A well-defined assessment produces more than a list of possible vulnerabilities. It establishes what was examined, under which conditions, using which methods, and with what limitations. That foundation allows system owners to understand the significance of the findings and provides a defensible basis for remediation decisions.

# Combining Code Review, Protocol Analysis, and Configuration Auditing

No single testing method can provide a complete view of drone security. Source-code review can reveal unsafe assumptions and implementation errors, protocol analysis can identify weaknesses in communication behavior, and configuration auditing can expose protections that are missing or incorrectly enabled. Each method examines a different aspect of the system. Combining them allows an assessor to compare the intended design with the actual implementation and determine whether separate weaknesses reinforce one another.

Code review examines software logic, data handling, access control, error handling, and the boundaries between components. In a flight-control system, relevant areas may include command parsing, mission validation, sensor-data processing, update verification, authentication routines, and failsafe logic. In a ground-control application, the focus may include credential handling, local storage, mission-file processing, network requests, and the enforcement of operator permissions. The objective is to understand how the program behaves when it receives ordinary input, malformed input, unexpected states, or requests from entities that should not possess the requested authority.

Security-sensitive code deserves particular attention because a small implementation error can undermine a larger design. An application may perform authentication but fail to check authorization before executing a privileged operation. A firmware update routine may verify the integrity of an archive but fail to authenticate its origin. A mission parser may validate individual fields but neglect the relationship between a command and the aircraft's current operating state. These problems illustrate why code review must consider complete execution paths rather than isolated functions.

Input validation is an important subject of review. Software frequently receives information from external sources, including mission files, telemetry messages, operator commands, configuration records, and network services. The implementation should establish expected formats, data types, value ranges, and contextual restrictions before acting on that information. Validation should occur at the component that uses the data, not only in an upstream application that may be bypassed or compromised.

Memory safety and resource management also matter in embedded systems. Depending on the programming language and implementation, incorrect bounds checking, invalid pointer use, unsafe memory handling, or poorly managed resources can cause crashes or unpredictable behavior. In a drone, such failures may affect communication availability, sensor processing, or other time-sensitive functions. Code review should consider whether errors are handled in a way that preserves system stability and prevents an ordinary input-processing failure from affecting unrelated critical functions.

Concurrency creates additional challenges. A flight controller or companion computer may process telemetry, sensor updates, mission commands, and maintenance requests concurrently. Shared state can be modified while another component is using it, producing inconsistent values or unexpected transitions. Reviewers should examine synchronization, task priorities, resource ownership, and state transitions where they affect security or safety. A race condition that appears harmless in ordinary operation may become significant when it affects authorization, configuration updates, or the sequence of safety checks.

Protocol analysis examines how components exchange messages and establish trust. Drone systems may use proprietary or open communication formats for telemetry, commands, device management, and payload operation. Regardless of the protocol, the

assessment should establish how messages are framed, authenticated, validated, ordered, and associated with a particular session or device. It should also examine how the system responds to missing fields, invalid values, repeated messages, unexpected message types, and communication interruptions.

Encryption alone does not establish that a protocol is secure. A system may encrypt traffic but fail to authenticate the communicating party, allowing an unauthorized participant to establish a connection. It may authenticate a session but accept commands that lack appropriate authorization. It may protect message integrity but fail to detect replay of an old, previously valid instruction. Protocol review must therefore examine the complete security design, including identity verification, message authentication, freshness, session management, and authorization.

The interpretation of protocol fields deserves close attention. If different components interpret a field differently, the same message may produce inconsistent behavior across the system. A ground station may believe that a command represents a harmless configuration request, while the aircraft interprets it as a more consequential operation. Clear protocol specifications, strict validation, and consistent handling of unsupported values reduce this risk. Where behavior is undocumented, the assessor should distinguish an observed implementation detail from an established protocol requirement.

Configuration auditing complements code and protocol review by examining the settings that determine how protections operate in practice. A device may support secure boot, but the relevant feature may not be enabled. A communication service may require authentication in one configuration but permit unrestricted access in another. Debugging interfaces may be restricted during development but remain available on production hardware. These are configuration questions that cannot always be resolved by reading source code alone.

The audit should compare the deployed or test configuration against an approved baseline. Relevant settings may include access permissions, authentication requirements, cryptographic policies, firmware verification options, network exposure, logging behavior, update restrictions, and safety-related parameters. Differences should be evaluated in context because not every deviation is a vulnerability. Some systems require different configurations for development, maintenance, and production, and the security question is whether each configuration is appropriate to its purpose.

The strongest assessments correlate evidence from these methods. Suppose code review identifies a command handler that does not independently check the operator's authorization. Protocol analysis may establish that the relevant message can reach the handler through a particular service, while configuration auditing determines whether that service is exposed in the deployed configuration. Together, these findings provide a more complete understanding of the risk than any one observation could provide alone.

Similarly, a review of firmware-update code may reveal that package authenticity depends on a configuration setting. An audit can establish whether the setting is enabled on the test device, while release documentation can determine whether the intended production process configures it consistently. This combination helps distinguish a theoretical implementation concern from a verified weakness in the deployed system.

Static analysis tools can assist code review by identifying patterns associated with certain classes of defects. Dynamic analysis can reveal behavior that occurs during execution. Protocol analyzers can help interpret message exchanges, and configuration-audit tools can compare settings with established requirements. These tools are useful aids, but their output requires expert interpretation. Automated findings may be false positives, miss contextual

conditions, or fail to recognize that a particular control is enforced elsewhere in the architecture.

A disciplined reviewer records both the evidence and the assumptions behind the finding. The report should identify the relevant component, the expected behavior, the observed behavior, and the conditions under which the discrepancy occurs. It should also explain what remains unverified. This precision helps engineering teams reproduce the problem and prevents a limited observation from being presented as proof of a broader compromise.

## Fuzz Testing and Robustness Analysis in Isolated Environments

Fuzz testing is a software-testing technique that supplies a program with a broad range of inputs to discover unexpected behavior. Rather than relying only on manually selected test cases, a fuzzer generates or mutates inputs and observes whether the target program crashes, violates defined properties, hangs, consumes excessive resources, or behaves inconsistently. This method is particularly useful for software that processes complex external data, including mission files, communication messages, configuration records, firmware packages, and sensor-related inputs.

The value of fuzz testing comes from the difficulty of anticipating every possible input combination. A parser may handle ordinary messages correctly but fail when fields are missing, values exceed expected ranges, records appear in an unusual order, or lengths are inconsistent. A mission file may be valid in most respects but contain an unusual combination of commands that exposes a faulty state transition. Fuzz testing can uncover such conditions more efficiently than relying entirely on hand-written examples.

Testing should begin with a clearly defined target. The target may be a standalone parser, a communication library, a mission-validation component, or a firmware-update handler. Narrow targets are often easier to investigate because failures can be connected to a smaller amount of code. Where the component is safety-critical, the test environment should be isolated from live flight operations and configured to prevent generated inputs from reaching operational aircraft.

A useful starting point is a collection of representative valid inputs. These examples establish normal behavior and help the fuzzer generate mutations that remain close enough to the expected format to exercise meaningful code paths. For a mission parser, the initial corpus might contain approved sample missions with different waypoint counts, supported command types, and valid configuration combinations. For a protocol parser, it may contain documented message types and examples of ordinary exchanges captured in an authorized test environment.

Mutation-based fuzzing changes existing inputs by altering values, lengths, field order, or other structural features. Generation-based fuzzing constructs inputs according to a model of the expected format. Both approaches have value. Mutation can be effective when valid samples are available, while generation can provide broader control over structured formats. The choice depends on the target, the complexity of the input, and the available testing tools.

A fuzzer is most useful when paired with a clear definition of failure. A process crash is an obvious signal, but many security defects do not produce immediate crashes. A parser may accept a value outside its documented range, a command handler may enter an invalid state, or a validation routine may silently discard a required restriction. Tests should therefore check relevant properties as well as basic process stability.

For example, a mission-validation component should consistently reject malformed or unsupported input without accidentally producing an executable mission. A configuration parser should not accept invalid values merely because they can be converted into the expected data type. A communication service should preserve its authorization requirements when it receives incomplete or unexpected messages. These properties can be tested using assertions, expected outcomes, and automated checks that compare the observed behavior with the defined requirements.

Robustness analysis examines how software behaves under abnormal but plausible conditions. These conditions may include truncated messages, delayed responses, interrupted file transfers, repeated requests, unexpected sensor values, exhausted memory, or temporary loss of a dependent service. The aim is not merely to identify crashes but to determine whether the system remains within its defined operating limits when normal assumptions no longer hold.

Resource exhaustion is an important part of this work. Embedded systems have limited memory, processing capacity, power, and communication bandwidth. An input that triggers excessive allocation or repeated processing may affect other functions even if the program does not crash. Testing should therefore examine whether the target enforces reasonable limits on message sizes, nesting depth, processing time, queue length, and the number of concurrent requests, as appropriate to the component.

Timing behavior requires particular care in real-time systems. A delay that is inconsequential in a desktop application may interfere with a time-sensitive embedded task. However, a test environment can produce timing characteristics that differ substantially from actual flight hardware. The assessor should therefore use simulation and controlled hardware testing to establish what the results mean. A timeout observed in a virtual environment may identify a useful

defect, but its operational significance must be assessed against the real system's timing requirements.

Fuzz testing can also examine interactions between components. A message may pass through a communication parser, a validation layer, and a command handler before it affects system state. Testing each component independently may miss defects that arise from inconsistent assumptions between them. Integration testing can reveal cases in which one component accepts a value that another interprets differently. These tests should remain bounded and should avoid transmitting experimental inputs to operational aircraft.

Coverage information can help assess how thoroughly a test campaign explores the target. Code coverage indicates which portions of the program were executed, while other measures may track branches, states, or protocol transitions. High coverage does not prove that the software is secure, and a low crash count does not establish robustness. Coverage is best used to identify under-tested areas and improve the quality of the test corpus.

When a failure occurs, the original input should be preserved. The assessor should record the software build, configuration, test environment, and relevant execution details. A minimized input that retains the failure is often easier to analyze than the original complex case. Minimization removes unnecessary parts of the input while preserving the observed behavior, helping engineers identify the smallest condition that triggers the problem.

Isolation is essential because fuzzing may expose unstable behavior. Tests should use dedicated processes, virtual machines, containers, emulators, or hardware-in-the-loop setups as appropriate. These controls help prevent a malformed input from affecting unrelated systems and make it easier to reset the target between tests. For hardware testing, appropriate power protection and recovery procedures should be available before the campaign begins.

Fuzz testing has limitations. It may fail to reach a rarely executed code path, may generate inputs that are rejected before reaching the relevant logic, or may miss defects that require a specific environmental condition. It can also produce many failures that have no security significance. The results should therefore be combined with code review, protocol analysis, requirement-based testing, and expert assessment of the affected behavior.

## Reproducing Findings and Distinguishing Defects From Security Vulnerabilities

A potential security finding becomes useful when it can be explained, reproduced, and connected to a credible consequence. During an assessment, an unexpected result may initially appear significant but later prove to be a test-environment problem, a documented limitation, or an ordinary software defect without meaningful security impact. Conversely, a seemingly minor inconsistency may reveal a failure in an important security boundary. Careful reproduction and classification prevent both overstatement and underestimation.

Reproduction begins by preserving the conditions under which the behavior occurred. The assessor should record the relevant software version, hardware revision, configuration, input, test procedure, and observed outcome. Where the finding depends on a sequence of events, that sequence should be documented clearly enough for another qualified engineer to repeat it. If the result occurred only once, the report should state that limitation rather than presenting it as a consistently demonstrated behavior.

A minimal reproduction is usually preferable to a complicated demonstration. It should include only the steps and conditions

necessary to trigger the behavior. This makes the finding easier to understand and helps isolate the relevant component. When a large mission file or complex communication exchange is involved, reducing the input while preserving the result can reveal which field, command sequence, or state transition is responsible.

The test environment must be considered during reproduction. A failure may result from a missing dependency, an incorrect configuration, inadequate resources, or a difference between simulated and physical hardware. The assessor should determine whether the behavior persists when the test is repeated under the documented conditions and whether it appears on a second equivalent environment where practical. Differences between environments should be recorded because they may explain why a weakness affects only certain deployments.

A software defect is not automatically a security vulnerability. Defects include incorrect calculations, poor usability, unsupported features, reliability problems, and violations of functional requirements. A security vulnerability is a weakness that can be used or triggered in a way that undermines a security property or creates a meaningful safety or operational consequence within the relevant threat model. The distinction depends on behavior, exposure, authority, and impact rather than on the mere presence of a programming error.

For instance, a user interface may display an incorrect status message while the underlying aircraft state remains correct. This is a defect that may still deserve correction, especially if it misleads an operator, but its security significance depends on the operational context. If the same reporting error conceals a disabled geofence or causes an operator to believe that a security restriction is active when it is not, the issue may have a more direct safety and security consequence.

A crash also requires contextual evaluation. A parser that crashes when given malformed data may present a denial-of-service risk if an unauthorized party can deliver that data to an exposed service. If the same parser is available only in an isolated offline development tool and cannot affect a deployed system, the operational risk may be much lower. The technical defect may be identical, but the threat model and practical impact differ.

Reachability is an important part of the analysis. The assessor should determine whether the affected code can be reached under the deployed configuration and whether an attacker possesses a plausible route to supply the triggering input. A weakness in unused code may still matter if future configurations activate it, but that concern should be distinguished from a weakness that can be reached through a currently exposed interface. Reports should identify the conditions necessary for the behavior to occur.

Privileges also affect classification. If a function can be invoked only by a fully authorized administrator, a weakness may have a different risk profile from one that can be triggered by an unauthenticated network participant. However, privileged access does not automatically make a vulnerability unimportant. An attacker who compromises an ordinary operator account may be able to exploit a flaw in a privileged service, and an insider may misuse access that was legitimately granted. The analysis should account for realistic paths from an initial position to the affected function.

Security boundaries provide another useful basis for classification. A defect may become a vulnerability when it permits one component to exercise authority that should belong to another. Examples include a payload application changing flight-critical settings without authorization, a maintenance service exposing credentials to an ordinary user, or a mission validator accepting commands outside the approved policy. The central issue is whether the implementation

violates an intended security boundary and whether that violation has a credible consequence.

False positives must be handled systematically. Automated tools may flag insecure patterns that are unreachable, mitigated elsewhere, or irrelevant to the particular build. A configuration scanner may report a missing control that is implemented through a different approved mechanism. The assessor should investigate the finding, inspect the surrounding design, and seek corroborating evidence before assigning a final classification. Dismissing every uncertain result is as unhelpful as accepting every automated alert without review.

Reproducibility does not require unnecessarily demonstrating the most damaging possible outcome. If a controlled test establishes that an unauthorized input bypasses a critical validation check, it may be sufficient to document that behavior and its implications. There is no general need to progress to a live flight disruption, exposure of real credentials, or permanent alteration of operational equipment when safer evidence can establish the same weakness. The assessment should use the least intrusive test that provides adequate confidence.

Findings should distinguish observed facts from inferred consequences. An assessor may directly observe that a command is accepted without an expected authorization check. The conclusion that this could allow unauthorized mission changes is an impact assessment that depends on the command's capabilities and the surrounding controls. The report should explain the relationship between the observation and the conclusion so that reviewers can evaluate the reasoning.

A useful finding includes enough technical detail to support remediation without exposing unnecessary sensitive information. It should identify the affected component and version, explain the required conditions, describe the observed behavior, and state the expected behavior. It should also identify any uncertainty, environmental limitation, or mitigating control that affects the

result. Where appropriate, a controlled test case can be retained so that engineers can verify the correction later.

Remediation verification closes the assessment loop. After the responsible team implements a correction, the assessor should repeat the original test and confirm that the problematic behavior no longer occurs. Related tests should also be performed to ensure that the correction has not weakened another control or disrupted legitimate operation. A patch that prevents one malformed message but causes valid mission files to be rejected may resolve one defect while creating another operational problem.

## Prioritizing Risks by Exploitability, Safety Impact, and Operational Consequences

Security assessments often identify more weaknesses than an organization can correct immediately. Risk prioritization helps determine which findings require urgent action, which should be scheduled for planned remediation, and which can be accepted temporarily with documented justification. In drone systems, this decision must consider not only conventional cybersecurity outcomes but also the potential effects on flight safety, mission reliability, equipment, personnel, and surrounding infrastructure.

Severity and risk are related but distinct. Severity describes the seriousness of a vulnerability under defined assumptions, while risk reflects the likelihood and consequences of the vulnerability affecting a particular system in its operating environment. A severe weakness may present limited immediate risk if the affected service is not deployed or cannot be reached under the current configuration. A less technically complex flaw may present substantial risk if it affects an exposed, safety-critical function used during every mission.

Exploitability concerns the practical conditions required to take advantage of a weakness. The assessment should consider whether an attacker needs physical access, an authenticated account, a particular network position, specialized equipment, or detailed knowledge of the system. It should also consider the complexity of the required conditions and whether those conditions are realistic in the aircraft's operating environment. These factors should be based on evidence and a documented threat model rather than assumptions about what an attacker might theoretically achieve.

Exposure affects the analysis. A service available only through a controlled maintenance connection has a different attack surface from a service reachable through a broadly accessible network. An interface that remains physically exposed during routine deployment may create a different risk from one accessible only during supervised laboratory work. Exposure should be evaluated alongside the strength of authentication, the privileges available through the interface, and the controls that limit the effects of misuse.

Potential impact must be assessed across multiple security properties. A vulnerability may expose sensitive imagery or mission records, permit unauthorized changes to configuration, interrupt telemetry, or affect the availability of an onboard service. In a cyber-physical system, it may also affect navigation, flight-control behavior, or the execution of safety restrictions. These consequences should be described separately where necessary because a single technical issue can affect several aspects of system operation.

Safety impact deserves particular attention. An interruption to a noncritical reporting service may have little direct effect on aircraft control, although it could impair maintenance or incident investigation. A similar failure in a service that supplies critical sensor information may have much greater consequences. The assessor should examine the role of the affected component, the system's redundancy, its failure-detection capability, and the

behavior that follows when the component becomes unavailable or untrustworthy.

The relationship between cybersecurity and safety is not always direct. An authentication failure may allow unauthorized access without immediately affecting flight behavior. Its consequences may depend on which commands are exposed, whether additional safety checks apply, and whether the aircraft is airborne or on the ground. A risk assessment should follow the plausible sequence from the initial weakness to the eventual outcome rather than assigning the highest impact simply because the system is an aircraft.

Operational consequences include more than physical harm. A vulnerability may cause mission cancellation, loss of collected data, equipment damage, regulatory difficulties, expensive maintenance, or interruption of a service on which an organization depends. For commercial operations, these consequences may affect contractual obligations and customer confidence. For public-safety or emergency operations, an interruption may reduce the availability of a service at a critical time. The relevant impact depends on the mission and the organization responsible for operating the system.

Environmental context can change the priority of a finding. An aircraft operating in a controlled test area may present a different consequence profile from one conducting inspection near people, traffic, or critical infrastructure. The same unauthorized mission change may be manageable when the aircraft is secured on a workbench but serious when the aircraft is operating remotely. Risk assessments should state the relevant operational assumptions and avoid treating all deployments as equivalent.

Existing safeguards can reduce risk, but their effectiveness must be demonstrated. A geofence may limit the effect of an unauthorized route change, provided it is correctly configured and enforced independently. Authentication may restrict access to a vulnerable service, provided the authentication mechanism itself is sound and

the service cannot be reached through another route. Redundancy may reduce the effect of a component failure, provided the redundant components do not share the same weakness. Mitigations should be credited only to the extent supported by evidence.

Common vulnerability-scoring methods can provide a consistent starting point for describing technical severity. However, a numerical score should not replace an assessment of the aircraft's actual operating conditions. A generic score may not fully represent the safety impact of a navigation-related defect or the operational significance of a vulnerability in a particular fleet. Organizations should use standardized scoring where appropriate while documenting drone-specific factors that change the priority.

Risk prioritization should also consider the scope of affected equipment. A vulnerability confined to a single development unit may require a different response from one present across an entire fleet. If multiple aircraft share the same firmware, cryptographic credential, or software dependency, one weakness may have a much wider potential impact. Software inventories, build provenance, hardware records, and fleet configuration data help determine how many devices may be affected and how quickly remediation must occur.

The availability of a reliable fix is another consideration, although it should not be used to dismiss serious findings. A vulnerability may require an immediate operational restriction while engineers develop and validate a permanent correction. Temporary measures could include disabling a nonessential service, limiting access to a maintenance interface, suspending a particular operation, or increasing monitoring. The suitability of each measure depends on whether it actually reduces exposure without creating a greater safety or availability problem.

Remediation priorities should be documented and assigned to responsible owners. High-risk findings may require immediate

containment, a defined remediation deadline, and explicit approval before affected systems return to operation. Lower-risk findings may be addressed through a scheduled software release or hardware revision. If an organization accepts residual risk temporarily, the decision should identify the rationale, applicable conditions, compensating controls, approval authority, and date for reassessment.

Prioritization is not a one-time decision. A vulnerability that appears manageable may become more serious after a new service is exposed, an aircraft begins operating in a different environment, or a previously trusted dependency is found to be compromised. Conversely, a well-validated correction or a substantial reduction in exposure may lower the residual risk. The assessment should be updated when new evidence changes the likelihood or consequence of exploitation.

The final objective of risk-based validation is to help engineering and operational teams make defensible decisions. A finding should be prioritized according to the evidence about its exploitability, the importance of the affected asset, the effectiveness of existing controls, and the consequences that could follow. In unmanned aircraft, this approach ensures that cybersecurity decisions account for the system's physical responsibilities as well as its digital assets, allowing limited remediation resources to be directed toward the weaknesses that pose the greatest credible threat to safe and reliable operation.

# Drone Defense, System Hardening, and Incident Response

## Applying Defense in Depth Across Hardware, Software, and Communications

Securing an unmanned aircraft requires more than protecting a single device or installing a security application on its onboard computer. A drone operates as an interconnected system in which flight-control electronics, firmware, navigation sensors, communication links, ground control software, operator credentials, and supporting infrastructure depend on one another. A weakness in any of these components can affect the reliability of the entire platform. Effective defense therefore begins with an architectural understanding of how information, authority, and safety-critical decisions move through the system.

Defense in depth is the practice of establishing multiple complementary safeguards so that the failure of one control does not automatically expose the entire system. For drone security, this principle must account for both conventional cybersecurity risks and the physical consequences of a compromised or malfunctioning aircraft. An unauthorized change to a ground application may initially appear to be an information technology problem, but if that application can submit mission commands, alter navigation settings, or change operational limits, the consequences may extend into

physical safety. Security controls must consequently be designed around the privileges each component possesses and the damage that could occur if those privileges were misused.

A well-designed architecture begins by separating flight-critical functions from less sensitive computing activities. The flight controller should retain authority over essential stabilization, actuator control, and predefined safety responses without depending unnecessarily on applications responsible for mapping, video processing, cloud connectivity, or data analytics. A companion computer may require access to mission information or selected flight commands, but it should not automatically receive unrestricted authority over every flight-critical parameter. Where the hardware and software support it, explicit interfaces should define which commands are permitted, which data can be read, and which functions remain under the exclusive control of the flight controller.

The same principle applies to the ground station and remote services. A ground application may authenticate an operator, display telemetry, prepare missions, and communicate with the aircraft. However, a compromise of the operator's ordinary computer should not grant an attacker unlimited access to every aircraft function, stored credential, and administrative service. Separate privileges, narrowly defined interfaces, and independent verification of sensitive actions can reduce the extent to which a single compromised component affects the rest of the system.

Hardware security forms an important part of this architecture. Physical access may allow an unauthorized person to disconnect components, attach diagnostic equipment, remove storage media, or attempt to alter firmware. Secure enclosures, controlled access to maintenance connectors, protected storage, and appropriate restrictions on debug interfaces can make such activity more difficult. These measures should reflect the operating environment. An aircraft used in a controlled research facility faces different

physical risks from one deployed in remote industrial locations, agricultural areas, or public infrastructure inspections.

Software protections must complement these physical measures. Secure configuration, verified software releases, application isolation, restricted administrative privileges, and disciplined patch management help prevent individual weaknesses from becoming system-wide failures. Communications protection adds another layer by verifying the identity of communicating parties, protecting sensitive information against unauthorized disclosure, and rejecting commands that do not satisfy established authorization rules.

The layers should not merely exist independently; they should reinforce one another. Encryption, for example, can protect a communication channel against passive observation, but it does not necessarily prevent an authenticated account from issuing an inappropriate command. Authentication can establish which device or operator is communicating, but it does not establish that every requested action is safe. Authorization can limit the available actions, while independent flight-envelope checks can reject commands that violate operational constraints. Each mechanism addresses a different aspect of the risk.

A further requirement is resilience. A drone should not lose every protective capability when its internet connection becomes unavailable, a cloud service fails, or a companion application stops responding. Critical safety decisions should be made locally whenever practicable, using validated onboard information and predetermined operating rules. A communications failure should trigger a defined response rather than an improvised reaction. Depending on the aircraft, mission, environmental conditions, and approved operating procedures, that response might involve holding position, returning to a designated location, landing, or entering another carefully evaluated safe state. No single response is appropriate for every aircraft or circumstance.

Defense in depth also depends on understanding trust boundaries. These boundaries separate components that have different security responsibilities or levels of authority. A payload processor, for instance, may handle camera data but should not automatically be trusted to modify flight-controller firmware. A remote analytics service may receive selected telemetry but should not need unrestricted access to operator credentials. A maintenance laptop may be authorized to install approved software during a service session but should not remain permanently trusted after the work is complete.

These boundaries must be reflected in technical controls and operating procedures. Network segmentation can limit communication between onboard services. Application permissions can restrict access to sensitive files and devices. Separate accounts can distinguish ordinary operation from maintenance and administration. Physical access policies can determine who may connect equipment to diagnostic interfaces. Monitoring can reveal attempts to cross boundaries that should remain closed.

Risk assessment determines how much protection each layer requires. Engineers should identify assets, possible attackers, operational dependencies, and credible failure scenarios before selecting controls. The loss of nonessential video metadata may have limited consequences, whereas unauthorized control of flight surfaces or alteration of navigation parameters can threaten the aircraft and people nearby. Security investment should reflect these differences without neglecting less obvious dependencies, such as the software used to prepare missions or the computers that distribute firmware updates.

A practical defense-in-depth strategy therefore combines prevention, detection, containment, and recovery. Prevention makes unauthorized actions more difficult. Detection identifies suspicious activity or unexpected changes. Containment limits the

consequences when preventive measures fail. Recovery restores a trustworthy operating condition without introducing additional hazards. Treating these functions as parts of one security architecture allows a drone operator or engineering team to evaluate not only whether individual controls exist, but whether the complete system can withstand realistic failures and recover safely.

## Implementing Secure Boot, Signed Updates, and Configuration Protection

The integrity of the software running on an aircraft is fundamental to its security. A drone may have strong communications encryption and carefully managed operator accounts, yet those protections become less meaningful if an attacker can replace its firmware or alter critical configuration values. Software integrity controls address this problem by establishing a verifiable basis for deciding which code and settings the system should trust.

Secure boot is a mechanism that checks the authenticity or integrity of software during startup before allowing the relevant code to execute. A typical implementation begins with a protected root of trust, such as immutable code in a microcontroller or another hardware-supported verification mechanism. That initial component verifies the next stage of the boot process, which in turn verifies subsequent components according to the platform's design. This chain continues until the operating system, flight-control software, or other protected application reaches an accepted state.

The security of secure boot depends on the trustworthiness of the initial verification mechanism. If an attacker can freely replace the code responsible for checking signatures, the remainder of the chain may offer little protection. Likewise, if a device accepts software signed by an unrestricted or compromised signing key, the presence

of signature verification alone does not establish meaningful security. The complete design must consider the protection of verification code, the authority represented by signing keys, and the rules governing which software versions are permitted to run.

Digital signatures help establish software authenticity. A software publisher signs a release using a private cryptographic key, and the receiving device verifies that signature using the corresponding trusted public key. When implemented correctly, this process provides evidence that the signed package was produced by an entity controlling the authorized signing key and that the signed content has not changed since signing. It does not independently prove that the software is free of defects, safe in every operating condition, or suitable for a particular aircraft. Those questions require testing, review, and controlled release procedures.

A secure update mechanism should verify packages before installation and reject unsigned, improperly signed, corrupted, or otherwise unauthorized releases. The verification process should cover not only the main flight-control binary but also relevant boot components, configuration schemas, onboard applications, and other executable elements that influence system behavior. The exact scope depends on the platform and the consequences of compromising each component.

Update installation must also account for interruption and failure. An aircraft that loses power halfway through a firmware update may be left unable to boot or operate safely if the update mechanism is poorly designed. Suitable systems may use protected recovery modes, redundant firmware partitions, staged installation, or another validated method that preserves a recoverable state. These mechanisms should be tested under realistic failure conditions rather than assumed to work because they are present in the design.

Rollback protection is another important consideration. A valid digital signature does not necessarily mean that a software version is

current or secure. An older release may contain a vulnerability that has since been corrected. If an attacker can force the aircraft to install or boot that older version, the system may return to a known vulnerable condition despite correctly verifying its signature. Version policies, protected counters, release metadata, and controlled recovery procedures can help prevent unauthorized downgrades. Nevertheless, rollback restrictions must be balanced against the need to recover from a defective release. Emergency recovery should remain possible through an authenticated, documented, and carefully controlled process.

Signing-key management deserves particular attention. Private signing keys should not be stored casually on general-purpose computers or shared among personnel without clear accountability. Organizations should restrict access to authorized release personnel, protect signing infrastructure, separate development and production permissions, and retain records of signing activity. Where appropriate, dedicated cryptographic hardware or managed key-protection services can reduce exposure. Key compromise procedures must specify how affected keys are revoked, how replacement trust material is distributed, and how deployed aircraft are returned to a trustworthy state.

Configuration protection is equally important because many security failures arise from changes to settings rather than replacement of the underlying software. Flight-controller parameters, geofence definitions, communication settings, authentication requirements, navigation thresholds, and fail-safe behavior can substantially affect an aircraft's behavior. A configuration file that is technically valid may still be dangerous if it disables an essential safeguard or changes a safety limit without authorization.

A robust configuration process begins with a controlled baseline. The baseline records approved settings for a particular aircraft model, hardware revision, software release, and operational purpose. Changes should be reviewed, tested, approved, and documented. The

organization should be able to determine which settings were active during a mission and who authorized any significant modification. Where technically feasible, configuration records can include cryptographic hashes or authenticated manifests that help reveal unexpected changes. A hash alone, however, only detects changes when the reference value is itself protected and trustworthy.

Not every configuration value should be treated identically. Some parameters may require adjustment for each mission, while others establish enduring safety or security requirements. A well-designed system distinguishes routine operational choices from restricted settings that require additional authorization. Changes to a mission's waypoints, for example, may be part of normal flight preparation, while changes to authentication policy, firmware verification, or critical fail-safe behavior should receive stronger protection.

Configuration management must include backups and restoration procedures. A backup is useful only if it is known to be trustworthy, compatible with the target software and hardware, and recoverable when needed. Restoring an outdated configuration without reviewing its contents can reintroduce a previously corrected weakness. Consequently, restoration should include integrity verification, compatibility checks, and a controlled review of security-sensitive settings.

Testing is essential before updates and configuration changes reach operational aircraft. Simulation can help identify unexpected behavior, while hardware-in-the-loop testing can evaluate interactions between flight software and representative hardware. Ground testing can verify boot integrity, communication functions, and recovery behavior before flight. Changes affecting flight-critical components should be introduced through approved maintenance procedures, with the aircraft kept out of service until required checks have passed.

Secure boot, signed updates, and configuration protection are most effective when supported by a coherent software lifecycle. Developers must know which components require verification, operators must know how approved releases are identified, and maintainers must have a reliable method for recovering systems that fail validation. When these responsibilities are clearly assigned, software integrity becomes an operational property rather than a feature that exists only in technical documentation.

## Strengthening Authentication, Encryption, and Access-Control Policies

Authentication establishes whether an operator, service, or device is the identity it claims to be. Authorization determines what that authenticated identity is permitted to do. Encryption protects information against unauthorized disclosure and, when incorporated into an appropriately authenticated protocol, can also contribute to protection against message modification and impersonation. These functions are related, but none should be treated as a substitute for the others.

Drone environments often contain several different identities. The aircraft itself may need to establish its identity to a ground station. An operator may need to prove authorization to control a particular aircraft. A maintenance technician may require access to diagnostic functions without permission to initiate a mission. A cloud service may need access to selected telemetry but not to flight-control commands. Effective security policies define these identities separately and grant each only the permissions necessary for its responsibilities.

Passwords alone may be insufficient for accounts that can modify safety-critical settings, authorize software updates, or control aircraft remotely. Multifactor authentication can reduce the risk that a stolen password will be enough to gain access, particularly for administrative accounts and remote management systems. The authentication method should be selected according to operational requirements, device capabilities, and the consequences of account compromise. Where connectivity is unreliable, organizations should plan secure alternatives that do not rely on uncontrolled bypasses or shared emergency passwords.

Device authentication is just as important as user authentication. A ground station should not trust every system that can reach a communication port, and an aircraft should not accept sensitive commands merely because they originate from a familiar network address. Network location can provide contextual information, but it is not a reliable identity credential on its own. Stronger designs use cryptographic identities or other platform-supported mechanisms to establish trust between communicating devices.

Mutual authentication is particularly valuable when both endpoints need assurance about the identity of the other. The aircraft can verify that a ground station is authorized, while the ground station verifies that it is communicating with the intended aircraft. Such arrangements reduce the opportunity for an unauthorized device to impersonate a legitimate endpoint. Their effectiveness depends on sound credential provisioning, certificate validation where applicable, secure key storage, and appropriate handling of expired or revoked credentials.

Encryption should protect sensitive communication links against passive interception and unauthorized modification. The specific protocol and cryptographic algorithms should follow current, well-reviewed standards appropriate to the device's capabilities and threat model. Custom cryptographic designs should be avoided

because subtle implementation errors can undermine otherwise sound intentions. Engineers must also verify that protection is actually enforced across every relevant communication path rather than assuming that one encrypted link secures the entire system.

A drone may communicate through several distinct channels, including a local radio connection, an onboard wireless network, a cellular connection, a satellite terminal, or an interface used during maintenance. Each channel may have different capabilities and limitations. Security policies should identify where authentication and encryption begin and end, what information remains exposed, and whether intermediary systems can read or alter the data. Protecting one channel does not automatically protect information after it reaches an untrusted application or service.

Access control translates security requirements into enforceable permissions. Role-based access control can assign privileges according to responsibilities such as pilot, mission planner, technician, security administrator, or software release manager. More detailed systems may also consider contextual factors, including the aircraft involved, the current operating state, the location of the request, or the sensitivity of the requested action. Regardless of the model, permissions should be specific enough to prevent routine accounts from accumulating unnecessary authority.

The principle of least privilege requires that users and services receive only the access needed to perform their assigned functions. A telemetry display does not normally require permission to change authentication settings. A video-processing application should not need access to signing keys. A maintenance account may require temporary diagnostic privileges but should not remain permanently elevated after the maintenance task ends. Removing unnecessary permissions reduces the number of pathways an attacker can exploit after compromising an account or application.

Privileged actions may warrant additional safeguards. Before a sensitive change is accepted, the system might require renewed authentication, confirmation by an authorized operator, or an independent policy check. The purpose is not to burden every ordinary action with repeated prompts. It is to introduce proportionate verification where the consequences of an error or unauthorized command are significant. Controls must also remain usable under realistic operating conditions; poorly designed security procedures can encourage operators to share credentials or bypass protections.

Credential lifecycle management is a continuing responsibility. Credentials must be generated securely, distributed through controlled processes, stored in protected locations, and replaced when their security may have been compromised. Personnel departures, lost devices, damaged equipment, and suspected unauthorized access should trigger defined revocation procedures. Organizations should know how to remove a device from the trusted fleet without creating an unsafe aircraft state or permanently losing the ability to perform legitimate recovery.

Shared accounts undermine accountability because they make it difficult to establish which individual performed an action. Whenever practicable, operators and administrators should use identifiable accounts, with records that connect sensitive changes to authenticated identities. Emergency accounts, if necessary, should be tightly restricted, monitored, and reviewed after use. Credentials should not be embedded in source code, distributed in ordinary configuration files, or reused across unrelated systems.

Authorization must also account for the difference between a technically permitted command and an operationally safe command. A validly authenticated pilot might request an altitude change that violates an established geofence or operating limit. The aircraft should apply independent safety checks appropriate to its design

rather than treating operator authentication as proof that every instruction is acceptable. Likewise, a valid command from a compromised application should not automatically bypass the flight controller's own limits.

These controls should be verified through routine testing. Teams can review permissions, examine expired and revoked credentials, validate encryption settings, confirm that unauthorized identities are rejected, and assess whether privileged actions generate appropriate audit records. Testing should take place on authorized equipment and within a defined scope. Findings should lead to specific corrective actions, with retesting used to confirm that the weakness has been addressed.

## Detecting Anomalous Behavior Through Telemetry, Logs, and System Monitoring

Preventive controls cannot eliminate every security risk. Software defects, compromised credentials, misconfigured systems, component failures, and unexpected environmental conditions may still produce abnormal behavior. Monitoring provides a means of recognizing these conditions early enough to investigate them, limit their consequences, and preserve evidence for later analysis.

Telemetry is particularly valuable because a drone continuously produces information about its operating state. Depending on the platform, this information may include position estimates, altitude, attitude, speed, battery condition, navigation quality, communication status, flight modes, sensor measurements, and system health indicators. Companion computers and ground stations may provide additional information about application activity, network connections, mission changes, and software processes. When interpreted together, these data sources can reveal

inconsistencies that would not be apparent from an individual measurement.

Effective monitoring begins with a clear understanding of normal behavior. A communication link may experience ordinary fluctuations in signal quality, and a navigation estimate may vary within expected tolerances. A processor may briefly increase its workload when processing imagery. Such variations do not automatically indicate an attack. The security team must distinguish expected operational variation from deviations that warrant investigation.

Baselines should reflect the aircraft's configuration and intended mission. A small inspection drone operating close to its controller has different communication patterns from an aircraft using a remote network connection. A mapping mission may generate substantial image-processing activity, whereas a simple waypoint flight may require fewer computational resources. Monitoring that ignores these differences can produce excessive false alarms or overlook unusual behavior that appears normal under an inappropriate baseline.

Useful indicators include repeated authentication failures, unexpected changes to flight modes, mission modifications outside an approved workflow, new network services, unexplained configuration changes, abnormal process activity, and discrepancies between independent sensor sources. A sudden loss of telemetry may be operationally significant, but it is not by itself proof of malicious interference. Environmental conditions, equipment faults, antenna problems, software failures, and network outages can produce similar symptoms. Investigators should therefore consider multiple explanations before assigning a cause.

Correlating events across components often provides stronger evidence than examining them in isolation. Suppose a ground station records a mission change that was not expected, the aircraft records

a corresponding parameter update, and the operator account logs show an unfamiliar login. Taken together, these events justify a more urgent investigation than any single record might warrant. Correlation can help reconstruct the sequence of events, identify the component through which the change occurred, and establish whether the event affected the aircraft's operation.

Time synchronization supports this process. If the flight controller, companion computer, ground station, and remote service use inconsistent clocks, it may be difficult to determine which event occurred first. Systems should maintain time accurately where practical and record the source and reliability of timestamps. When synchronization is unavailable, logs should preserve enough information to establish relative event order. Monitoring tools must also account for temporary loss of connectivity and delayed delivery of records.

Log integrity matters because an attacker with sufficient privileges may attempt to alter or delete evidence. Security-relevant records should be protected from unauthorized modification and access. Where supported, organizations can use restricted logging permissions, centralized collection, cryptographic integrity checks, or append-only storage. Logs held only on a compromised endpoint may be incomplete or unreliable, so important events should be copied to a protected location when communication and operational constraints permit.

Logging should be selective and purposeful. Recording every sensor sample or application event without considering storage, privacy, and review capacity can overwhelm the system. Instead, the organization should identify the events required to investigate security incidents, understand operational failures, and demonstrate compliance with internal procedures. These may include authentication attempts, privilege changes, firmware installation results, configuration updates, mission uploads, safety-state transitions, and changes to security monitoring itself.

Onboard monitoring introduces resource constraints. A drone may have limited processing power, memory, storage, and energy. Excessive logging or computationally expensive anomaly detection can interfere with mission performance or compete with flight-critical processes. Monitoring functions should therefore be designed and tested so that they do not compromise stabilization, control timing, or safety mechanisms. Where appropriate, less critical analysis can occur on a companion computer or ground system, while the flight controller retains its essential real-time responsibilities.

Anomaly detection can rely on fixed thresholds, rules, statistical models, or more sophisticated analytical techniques. Simple rules are often easier to understand and validate. For example, an alert can be generated when a restricted setting changes without an approved maintenance session. Statistical approaches may help identify unusual patterns that are difficult to express as fixed rules, but their output requires careful interpretation. A model trained on incomplete or unrepresentative data may produce false positives or miss meaningful events.

Automated detection should not be confused with automated authority to take every possible corrective action. A monitoring system might identify a suspected navigation anomaly, but it should not automatically initiate an aggressive maneuver without considering the aircraft's operating state and validated safety rules. Security responses that affect flight behavior must be evaluated as part of the aircraft's safety design. In some cases, the appropriate action is to alert the operator, preserve diagnostic data, or recommend a controlled transition to a predefined safe mode rather than attempt an unvalidated intervention.

Alert quality is central to effective monitoring. If the system repeatedly raises low-value warnings, operators may become accustomed to dismissing them. Alerts should communicate the

nature of the observed condition, its likely operational significance, and the evidence available to support further investigation. Where certainty is limited, the alert should state that the behavior is anomalous rather than declaring that an attack has occurred. Severity should reflect both the confidence of the detection and the potential consequences.

Monitoring also requires defined ownership. Someone must review alerts, distinguish routine faults from security incidents, and determine when escalation is necessary. For a small organization, these responsibilities may belong to a trained operator or maintenance lead. A larger fleet may require a centralized security function with established procedures for correlating events across aircraft. In either case, staff need sufficient training to interpret the evidence without making unsupported assumptions.

Privacy and data governance remain relevant. Telemetry and logs may reveal locations, mission schedules, operational patterns, personal information, or sensitive infrastructure details. Collection should be proportionate to legitimate security and safety needs, access should be restricted, and retention periods should be documented. Data shared with external service providers should be limited to what is necessary for the stated purpose and handled according to applicable legal and contractual requirements.

A monitoring program becomes more useful when its findings lead to improvements. Repeated communication failures may indicate an engineering issue, while recurring authentication failures may reveal weaknesses in credential management or operator training. Unexplained configuration changes may point to inadequate change control. Reviewing these patterns enables organizations to correct underlying causes rather than treating every alert as an isolated event.

# Developing Incident Response, Safe Recovery, and Security Maintenance Procedures

A drone security incident is any event that compromises, or credibly threatens to compromise, the confidentiality, integrity, availability, or authorized control of the aircraft and its supporting systems. Incidents may include unauthorized access, suspected command manipulation, malicious or unapproved firmware, exposed credentials, unexplained mission changes, compromised ground stations, or interference with systems essential to safe operation. They may also overlap with ordinary equipment failures, making careful assessment necessary before a definitive cause can be established.

Incident response should be designed before an event occurs. During an unexpected flight or system malfunction, operators may have only a short time to make decisions. Improvised security procedures can create additional risks, especially if an attempted investigation interferes with flight control or delays an established emergency response. A written plan should therefore define responsibilities, escalation paths, communication methods, evidence-handling requirements, and the conditions under which an aircraft must be removed from service.

The first priority is the safety of people and property. If an aircraft exhibits behavior that creates an immediate hazard, the operator should follow the approved emergency procedures for that aircraft and operating environment. Security investigations must not take precedence over necessary actions to prevent injury or damage. Depending on the circumstances, the safest response may involve a controlled landing, a predefined return procedure, or another established recovery action. The correct choice depends on the

aircraft's capabilities, the remaining control authority, the location, environmental conditions, and the applicable operating rules.

The response plan should distinguish between an aircraft that is still controllable and one whose control status is uncertain. If the system remains responsive, an operator may be able to move it into an approved safe state while limiting further exposure. If control has been lost or commands cannot be trusted, attempts to issue repeated or experimental commands may worsen the situation. Operators should use validated procedures and avoid untested actions that could produce unpredictable movement.

Once immediate safety concerns have been addressed, the organization should determine the scope of the incident. This includes identifying the affected aircraft, ground stations, operator accounts, communication services, software releases, and mission data. The investigation should establish what was observed, when it occurred, which controls may have failed, and whether other systems share the same exposure. A compromised maintenance laptop, for example, may present a broader risk if it has been used to service several aircraft.

Containment aims to prevent an incident from spreading or causing further harm. Depending on the findings, containment may require disabling a compromised account, revoking a device credential, isolating an affected ground station, restricting access to a remote service, or suspending use of a suspect software release. Actions must be coordinated with flight operations and maintenance personnel so that security measures do not inadvertently disable a necessary safety function. Disconnecting a system without understanding its role can be dangerous if the aircraft depends on that connection for an approved operating procedure.

An aircraft suspected of software compromise should not automatically return to normal service merely because it appears to function correctly. Its current software state, configuration, and

stored credentials may no longer be trustworthy. The organization should apply a documented evaluation process to determine which components need inspection, which credentials require revocation or replacement, and whether the aircraft can be restored using an approved recovery procedure.

Evidence preservation supports both technical investigation and accountability. Relevant records may include flight-controller logs, ground-station logs, authentication records, firmware version information, configuration files, mission plans, network events, maintenance history, and operator observations. Where practical, investigators should preserve original records, record when and how evidence was collected, and document any changes made during recovery. A verified copy can be used for analysis while the original is retained under appropriate protection.

Evidence collection should not compromise immediate safety. During an active emergency, restoring a safe condition is more important than capturing every diagnostic detail. Once the situation is stable, investigators can preserve available records before performing actions that might overwrite them, such as reinstalling firmware, resetting equipment, clearing logs, or restoring a configuration backup. If the system is suspected of being actively compromised, evidence-handling decisions should be coordinated with personnel qualified to assess the technical and operational risks.

Analysis should separate observed facts from hypotheses. A sudden change in heading, for example, could arise from a navigation problem, a sensor fault, a control-system defect, an environmental disturbance, or malicious activity. Investigators should compare telemetry, configuration history, software versions, communication records, and physical inspection findings before concluding that an attacker was responsible. Accurate incident reports should document uncertainty and identify what additional evidence would be required to resolve it.

Recovery means restoring a trustworthy and operationally safe condition, not merely restarting the aircraft. The process may include replacing compromised credentials, reinstalling approved firmware, verifying secure boot, restoring a validated configuration, patching affected software, and confirming that access-control policies remain effective. Any suspected hardware tampering should be examined before the equipment is returned to service. If the integrity of a critical component cannot be established, replacement may be more appropriate than attempting an uncertain repair.

Recovery should be validated through staged checks. These may begin with documentation and software verification, proceed to bench testing and communications checks, and then move to controlled functional testing under approved conditions. Where a defect could affect flight safety, required ground and flight checks should follow the manufacturer's procedures and the organization's maintenance requirements. A system should not be returned to operational use solely because the original symptom has disappeared.

Communication is another essential responsibility. Operators, technical staff, management, and relevant external parties may need different information during an incident. Internal reports should identify the affected assets, known operational impact, actions already taken, and decisions that remain outstanding. External notifications should follow applicable legal, regulatory, contractual, and organizational requirements. Reports should be accurate and proportionate, avoiding unsupported claims about attribution or impact.

Incident response plans should also address events involving sensitive information. If mission data, video recordings, credentials, or operator information may have been exposed, the organization should determine what information was involved, which systems received it, and whether unauthorized access can be substantiated.

Any required notifications should be handled through the appropriate legal and compliance channels. Data exposure should not be dismissed simply because the aircraft itself remained physically controllable.

After recovery, a structured review should identify why the incident occurred and whether the response was effective. The review should examine technical weaknesses, procedural gaps, training needs, communication problems, and assumptions that proved unreliable. Its purpose is to improve the system, not merely to assign blame. Corrective actions should have clear owners, deadlines, and verification criteria so that recommendations result in measurable changes.

Security maintenance supports this process by reducing the likelihood of recurrence. Aircraft, ground stations, companion computers, remote services, and maintenance equipment should be covered by a documented maintenance schedule. The schedule should identify approved software versions, review intervals, credential checks, configuration verification, log-retention requirements, and responsibilities for applying security updates. The exact frequency should reflect the system's exposure, operational criticality, update availability, and the risks associated with making changes.

Patch management requires careful judgment. Security updates may correct vulnerabilities, but they can also introduce compatibility problems or unexpected changes in flight behavior. Before deployment, updates should be reviewed for relevance, tested in representative environments, and approved through a defined process. Urgent vulnerabilities may justify accelerated action, but expedited procedures should still include risk assessment, validation, and a clear record of the decision. When a patch cannot be installed immediately, temporary mitigations should be documented and monitored until a permanent correction is available.

Maintenance must include the supporting infrastructure. A secure aircraft can still be exposed through an outdated operator laptop, an improperly protected network service, or a compromised software distribution process. Ground computers should receive appropriate updates, unnecessary services should be disabled, endpoint protections should be maintained, and administrative access should be restricted. Maintenance tools should themselves be treated as sensitive assets because they may possess privileges capable of changing aircraft software or configuration.

Inventory management is a prerequisite for consistent maintenance. Organizations should know which aircraft they operate, the hardware and firmware versions installed on each unit, which communication modules and payloads are attached, and which credentials or certificates are associated with the equipment. Without reliable inventory information, it is difficult to identify vulnerable devices, confirm patch coverage, or determine whether an affected component is present elsewhere in the fleet.

Retirement and disposal procedures also matter. An aircraft that is sold, transferred, or permanently removed from service may contain stored mission records, network credentials, certificates, wireless settings, or other sensitive information. Before transfer or disposal, the organization should follow an approved process for removing data, revoking credentials where appropriate, and confirming that the device no longer retains unauthorized access to operational services. Storage media that cannot be securely sanitized should be handled according to the organization's data-protection requirements.

Personnel training reinforces the technical controls. Operators should understand how to recognize suspicious behavior, when to stop a mission, how to report an incident, and which actions they must not attempt without authorization. Technicians should understand firmware verification, credential handling, configuration

control, and evidence preservation. Security personnel should understand the operational consequences of disabling or isolating aircraft components. Exercises can help these groups coordinate their responsibilities before a real incident places them under pressure.

Incident response should be tested periodically through controlled exercises. A tabletop exercise can examine decision-making after a suspected unauthorized mission change. A laboratory exercise can test whether a compromised ground station can be isolated without losing required records. A recovery exercise can verify that an aircraft can be restored from an approved software image and returned to a known configuration. Such exercises should use authorized equipment and safe test conditions, with scenarios designed to evaluate procedures rather than create unnecessary operational hazards.

The resulting maintenance and response framework should be proportionate to the size and complexity of the operation. A small team may maintain a concise set of controlled procedures, a verified asset inventory, and a clear escalation path. A larger fleet may require dedicated monitoring, formal incident coordination, centralized credential management, and documented assurance processes. In both cases, effectiveness depends on whether the procedures can be followed under actual operating conditions and whether the organization can demonstrate that identified weaknesses have been corrected.

# Integrated Drone Security Assessment and Professional Practice

## Developing an End-to-End Assessment Plan for a Representative Drone System

A comprehensive drone security assessment examines the aircraft as a complete operational system rather than treating its individual components as unrelated technical assets. Modern unmanned aircraft depend on interconnected hardware, embedded software, wireless communications, navigation systems, ground control applications, companion computers, remote services, and human operators. Each component introduces distinct security requirements, but the most significant weaknesses often emerge where these elements interact. An effective assessment must therefore establish how the system is constructed, how authority is distributed, how information moves between components, and how a compromise in one area could affect the reliability and safety of the entire platform.

Professional assessment begins with a clearly defined purpose. An organization may wish to evaluate an aircraft before purchasing it, verify the security of a newly developed platform, examine a fleet after a suspected incident, or determine whether recent software changes have introduced new vulnerabilities. These objectives influence the scope, testing methods, evidence requirements, and final recommendations. A procurement assessment may emphasize

supplier assurance, software provenance, update mechanisms, and long-term maintenance commitments. A development assessment may concentrate on architecture, code quality, protocol validation, and secure implementation. An incident-driven investigation may instead prioritize evidence preservation, affected assets, and the identification of unauthorized changes.

The assessment must also establish authorization before technical testing begins. Written authorization should identify the systems covered, the individuals permitted to conduct testing, the activities allowed, the testing environment, and the limits of acceptable disruption. This is particularly important for drones because radio transmissions, flight commands, navigation systems, and shared communication infrastructure can affect equipment or people beyond the immediate test environment. Ownership of an aircraft does not automatically confer permission to interfere with third-party networks, external services, or other aircraft operating nearby. The assessment should remain within its approved boundaries and comply with applicable aviation, communications, privacy, and cybersecurity requirements.

A representative drone system provides a useful basis for organizing the work. Consider an aircraft equipped with a flight controller, satellite-navigation receiver, wireless command link, companion computer, camera payload, ground control station, and optional remote connectivity. The assessment should document each component and its purpose, identify the interfaces connecting them, and establish which components can issue commands or change security-sensitive settings. This inventory becomes the foundation for understanding the system's attack surface and evaluating the consequences of a potential compromise.

The architecture should be documented at several levels. A physical view identifies processors, sensors, communication modules, storage devices, diagnostic connectors, and external peripherals. A software view identifies firmware, operating systems, applications, libraries,

update services, and configuration files. A communication view records the logical links between the aircraft, ground station, remote services, and maintenance equipment. An authority view clarifies which components can read information, issue commands, modify configuration, install software, or grant privileges. These perspectives complement one another and help prevent important trust relationships from being overlooked.

The assessment plan should distinguish safety-critical assets from supporting resources. Flight-control logic, actuator commands, navigation information, and fail-safe configuration can directly influence aircraft behavior. Operator credentials, cryptographic keys, mission files, and firmware packages may determine whether those functions remain under authorized control. Video recordings and routine telemetry may be less directly connected to flight safety, yet they can contain sensitive information or reveal operational patterns. Asset classification should reflect confidentiality, integrity, availability, and physical safety rather than relying exclusively on traditional information technology categories.

Threat modeling provides a disciplined way to translate architecture into testable questions. The assessor considers who might attempt unauthorized access, what capabilities that person might possess, which components are exposed, and what consequences could follow. Relevant scenarios include a compromised ground station, an untrusted software update, weakly protected diagnostic interfaces, inadequate authorization for mission changes, exposed remote services, or the misuse of legitimate operator credentials. The purpose is not to assume that every theoretical attack is practical. It is to identify credible pathways and determine which deserve the greatest testing effort.

The plan should define how each important risk will be evaluated. A security requirement might state that only approved firmware can execute, that sensitive commands require an authenticated and authorized source, or that a payload application cannot modify

restricted flight parameters. Each requirement should be connected to a verification method and an expected result. Firmware integrity may be assessed through documented boot-verification behavior and controlled tests using approved test images. Authorization rules may be examined through configuration review and tests involving accounts with different privileges. Network exposure may be evaluated through an inventory of listening services and an assessment of whether those services are necessary and appropriately protected.

Testing methods should match the level of risk. Static inspection and documentation review generally present less operational risk than live tests that alter aircraft behavior. Laboratory testing can provide greater control than testing a drone during an active mission. Where a test could interrupt communications, change configuration, corrupt storage, or affect flight behavior, the plan should specify appropriate safeguards and recovery arrangements. Destructive testing should not be performed on operational equipment unless explicitly authorized and justified by a controlled procedure.

A representative system should be assessed in a suitable environment, with simulation and hardware-in-the-loop facilities used where they can reproduce relevant behavior safely. Simulation can support examination of mission logic, state transitions, navigation anomalies, and recovery decisions. Hardware-in-the-loop testing adds interaction with representative electronics, allowing engineers to evaluate software responses under more realistic timing and interface conditions. Neither method proves that every real-world condition has been reproduced, so the final report should distinguish simulated findings from findings confirmed on physical equipment.

The plan must include operational constraints and stopping conditions. Assessors should know when testing must stop, who can authorize a restart, and how the system will be returned to a known state. Unexpected flight behavior, unexplained changes to safety-

critical configuration, evidence of an active compromise, or loss of reliable control should trigger the predefined response. This prevents a technical investigation from becoming an uncontrolled operational event.

A baseline should be established before testing. It should identify approved firmware versions, configuration settings, network services, accounts, cryptographic identities, and relevant hardware revisions. Existing defects should be documented so they can be distinguished from conditions introduced during assessment. Baseline records also allow assessors to determine whether a configuration has changed during testing and whether those changes were expected.

The final assessment plan should define deliverables, reporting standards, evidence-handling procedures, and retesting expectations. A professional engagement does not end when a vulnerability is discovered. Its purpose is to provide the system owner with defensible findings and practical guidance that supports risk reduction. The plan should therefore specify how results will be validated, how urgent findings will be communicated, how remediation will be tracked, and how unresolved risks will be handled when immediate correction is not feasible.

## Evaluating Firmware, Wireless Links, Protocols, and Network Exposure

Firmware assessment examines the software that directly supports a drone's hardware and operational functions. It may include boot code, flight-control applications, communication modules, sensor drivers, device-management components, and recovery mechanisms. Because firmware often executes with substantial privileges and operates close to physical hardware, defects can have consequences

that extend beyond ordinary application failures. The assessment should determine how firmware is obtained, verified, installed, configured, and maintained throughout the aircraft's service life.

The first stage is to establish the firmware's identity and provenance. Assessors should record the version, build information, hardware compatibility, supplier or development source, and documented security features. Where available, release manifests, checksums, digital signatures, and build records can help establish whether the installed image corresponds to an approved release. A checksum can reveal whether two files differ, but it does not independently establish authenticity unless the expected checksum comes from a trusted source. Digital signatures provide stronger evidence of origin when the signing keys and verification process are properly managed.

Static firmware analysis can identify potentially unsafe implementation patterns without requiring the software to execute. Depending on the available material and authorization, the review may examine embedded strings, configuration defaults, library versions, cryptographic implementations, authentication logic, update mechanisms, and references to diagnostic functions. The assessor should determine whether debugging features remain enabled in production builds, whether secrets are exposed in recoverable form, and whether security-sensitive decisions rely on untrusted input.

The presence of a suspicious string or an apparently dangerous function does not automatically establish a vulnerability. A diagnostic routine may be inaccessible in production, a library may have been patched by the vendor, or a configuration option may be disabled by a separate security control. Findings must therefore be validated against the actual software behavior and the relevant execution conditions. Where source code is available, code review can help clarify the purpose of a function and identify missing validation or inadequate privilege checks. Where only compiled

firmware is available, conclusions should reflect the limits of binary analysis.

Secure boot and update verification deserve particular attention. The assessment should establish whether the aircraft verifies software before execution, which components are included in the verification chain, how signing keys are protected, and how the system behaves when verification fails. Update tests should determine whether unauthorized or corrupted packages are rejected through approved laboratory procedures. The assessor should also examine recovery behavior, version restrictions, and the possibility that a valid but outdated release could restore a previously corrected weakness.

Wireless security requires an understanding of the aircraft's actual communication architecture. A drone may use a dedicated command link, a local wireless network, a cellular modem, or a satellite connection. Some systems combine several of these technologies, with each link supporting different traffic and security requirements. The assessment should identify which channels carry flight commands, telemetry, payload data, administrative access, or software updates. It should also establish whether communication is direct or passes through intermediary equipment and services.

The security of a wireless link depends on more than signal strength or the presence of encryption. Important questions include how communicating devices establish identity, whether sensitive messages are protected against modification, how authorization is enforced, whether session credentials can be reused improperly, and what happens when a link is interrupted. The assessor should examine whether the implementation follows documented security requirements and whether the system rejects invalid or unauthorized requests.

Wireless testing should be conducted in a controlled environment with appropriate permission and safeguards against interference. Initial examination can focus on configuration, protocol

documentation, approved diagnostic information, and passive observation of the test system's own communications. More intrusive tests should use dedicated equipment and an isolated setup designed to prevent unintended effects on other devices. The objective is to evaluate defensive properties, not to disrupt aircraft or communications systems outside the authorized scope.

Communication protocols require similar scrutiny. A protocol may define message formats and expected exchanges, but secure implementation also depends on authentication, authorization, validation, and state management. Assessors should examine how the system handles malformed messages, unexpected message sequences, missing fields, invalid values, repeated requests, and commands that arrive in an inappropriate operating state. Robust implementations should reject invalid input predictably and preserve safe behavior when a request cannot be processed.

A message that passes a basic format check should not automatically be accepted as a legitimate instruction. The receiving component must consider whether the sender is authorized, whether the requested operation is permitted, and whether the aircraft is in a state where the operation is safe. Protocol review should therefore examine both message syntax and the rules governing the meaning of each command. Particular attention is warranted where a communication interface can modify navigation settings, change flight modes, alter mission plans, or affect other safety-critical functions.

Network exposure assessment identifies the services and interfaces that can be reached from relevant network segments. A companion computer may expose administrative access, application programming interfaces, data-transfer services, telemetry relays, or diagnostic endpoints. Some may be necessary for operation, while others may remain enabled because of development practices or inherited software defaults. Each exposed service should have a

documented purpose, an assigned owner, and an appropriate security configuration.

The assessor should establish whether services are bound to the intended network interfaces, whether unnecessary ports are disabled, whether authentication is enforced, and whether access is restricted to approved sources. Network segmentation can limit the extent to which compromise of a payload application or remote service exposes flight-critical functions. However, segmentation should be verified rather than assumed effective because the architecture appears separated in documentation. The assessment must account for actual routes, firewall policies, service permissions, and permitted communication paths.

The ground control station is another significant part of the exposure assessment. It may store aircraft credentials, mission plans, telemetry, and software packages. It may also communicate with multiple aircraft or connect to enterprise networks. Assessors should review operating-system security, application privileges, update status, local account permissions, stored secrets, and the conditions under which the station can establish control sessions. If the ground station is compromised, its legitimate access may allow an attacker to affect several aircraft, making its security particularly important.

Remote connectivity adds dependencies that may fall outside the aircraft itself. Cloud-based fleet management, remote telemetry services, cellular gateways, and software distribution systems can introduce additional accounts, APIs, and trust relationships. The assessment should identify which services are essential, what information they receive, how they authenticate devices, and whether access to one service grants unnecessary authority over another. It should also examine what happens when a remote service is unavailable and whether essential safety functions remain available locally.

The resulting technical review should describe both verified weaknesses and relevant limitations. If the assessor cannot inspect a proprietary component, the report should identify that restriction rather than imply that the component is secure. If a test confirms that an unauthorized update is rejected, the finding should specify the tested version, configuration, and conditions. This level of precision makes the assessment useful for engineering decisions and avoids conclusions that exceed the available evidence.

## Correlating Technical Findings Across Interdependent Components

Individual security findings rarely provide a complete picture of system risk. A weak authentication policy on a companion computer may appear moderate when considered alone, and a permissive communication interface may appear similarly limited. If the companion computer can issue privileged flight commands through that interface, however, the combined weaknesses may create a more serious pathway to unauthorized control. Correlation is the process of examining how findings interact and determining whether their combined effect is greater than the risk suggested by each item independently.

The first requirement is an accurate model of dependencies. Assessors should understand which components exchange information, which services depend on external infrastructure, and which identities or permissions allow actions to cross trust boundaries. A data-flow diagram can show where telemetry travels and where mission commands are accepted. A trust-boundary diagram can clarify which components authenticate one another and where input crosses from a less trusted environment into a more privileged one. These models provide a common reference for comparing findings discovered by different testing methods.

Consider a system in which a ground station stores an operator credential, a companion computer receives mission information, and the flight controller accepts commands from that computer. A review may identify insufficient protection for the stored credential, broad permissions for a companion application, and inadequate validation of certain commands. The assessor should examine whether these weaknesses can be combined under realistic conditions, whether an independent control interrupts the pathway, and what consequences would follow if the pathway were completed.

Correlation must remain evidence-based. It is not sufficient to imagine a chain of weaknesses and declare that the entire chain is exploitable. Each transition must be supported by architecture, configuration, observed behavior, or a carefully justified technical analysis. Where a link in the chain remains unverified, the report should identify it as an assumption or an area requiring additional testing. This distinction helps system owners prioritize work without confusing theoretical possibility with demonstrated exposure.

Temporal correlation can be equally important. An unexpected firmware change may precede repeated authentication failures, altered mission parameters, or unusual network connections. The sequence may suggest that one event enabled another, but timestamps and event records must be examined carefully. Clock drift, delayed log collection, incomplete records, and unrelated faults can produce misleading sequences. Investigators should compare multiple sources and document the confidence attached to any inferred relationship.

Configuration dependencies often reveal risks that are not obvious from software review alone. A service may be designed to require authentication but run in a configuration that disables it. A network firewall may restrict access to a management interface, while a separate routing rule makes that interface reachable from an unintended segment. A firmware verification feature may be

implemented but not enforced for every component. Correlation connects the design intention to the deployed reality.

The same method applies to safety-related consequences. A navigation anomaly may not directly indicate a cybersecurity weakness, but the investigation should consider whether an unauthorized configuration change, unvalidated sensor input, or compromised data path contributed to the event. Conversely, a security alert may arise from a failing sensor or unstable communication link rather than deliberate activity. Bringing together flight telemetry, maintenance records, software versions, configuration history, and network observations can help distinguish these possibilities.

A useful technique is to construct a scenario-based attack or failure path for each high-priority concern. The scenario begins with a plausible initial condition, such as access to an exposed service or possession of a compromised account. It then identifies the permissions and interfaces that could enable subsequent actions, the safeguards expected to interrupt the path, and the operational consequences if those safeguards fail. This method supports both security analysis and safety review, particularly when a digital compromise could affect physical movement.

Assessors should also examine common-cause failures. Several safeguards may appear independent while relying on the same underlying component. For example, multiple applications may depend on one authentication service, or several aircraft may trust the same signing key. If that shared dependency is compromised, several layers may fail together. Genuine defense in depth requires attention to these shared dependencies, not simply a large number of controls.

Fleet-level correlation can identify patterns across multiple aircraft. A vulnerable software library may be present in several firmware versions, a maintenance process may repeatedly leave diagnostic

interfaces enabled, or a common ground station may store credentials for an entire fleet. A finding on one aircraft should therefore prompt a justified review of similarly configured systems. The scope of that review should be based on shared architecture, software provenance, and deployment records rather than an assumption that every aircraft is identical.

Risk analysis should reflect the complete pathway and the consequences of success. A technically difficult vulnerability may still deserve attention if exploitation would affect a safety-critical function. A simple weakness may present limited risk if effective isolation prevents access to sensitive assets. Assessors should evaluate the likelihood of the relevant conditions occurring, the privileges required, the reliability of the attack path, the presence of mitigating controls, and the potential effect on confidentiality, integrity, availability, and physical safety.

Correlation also helps avoid double-counting. Several findings may describe different symptoms of one underlying defect. If each is assigned a separate high-risk rating without considering their shared cause, the report may exaggerate the overall exposure. Conversely, grouping distinct weaknesses too broadly can conceal the different corrections needed to address them. The assessor should distinguish root causes, contributing conditions, and observable symptoms, then explain how they relate.

The resulting analysis should provide a coherent account of system risk. Rather than presenting disconnected observations, it should explain which components are involved, how trust or control crosses between them, which safeguards are effective or missing, and what the organization should change first. This system-level understanding allows engineering teams to prioritize remediation according to real exposure and operational impact.

# Producing Reproducible Evidence, Risk Ratings, and Remediation Reports

A professional security assessment must be supported by evidence that another qualified person can review and understand. A conclusion without adequate supporting records is difficult to verify, prioritize, or defend. Reproducibility does not mean that every test must produce identical results under every condition. It means that the methods, environment, observations, and limitations are documented well enough for another assessor to repeat the relevant procedure or independently evaluate the conclusion.

Evidence collection begins with a record of the test environment. Important details include the aircraft model and hardware revision, firmware and application versions, relevant configuration settings, test equipment, network arrangement, and the date of testing. Where a finding depends on a particular operating state, that state should be documented. Version information is especially important because a result obtained against one release may not apply to another.

Each finding should identify the observed behavior and the expected behavior. For example, a report may state that a service accepted a sensitive administrative request from an account that should have read-only privileges. The evidence should identify the applicable access-control requirement, the account role used during testing, the controlled conditions, the request category, and the observed result. Sensitive operational data should be redacted where necessary, but redaction must not remove the information required to understand the finding.

Screenshots, log excerpts, configuration records, test outputs, and relevant source-code references can support a finding when they are collected appropriately. Evidence should be captured close to the

time of testing and associated with a unique finding identifier. If records are transferred between systems or reviewers, integrity checks and controlled storage can help establish that the evidence has not been altered. A cryptographic hash may support this process, although it does not prove that the original evidence was accurate or complete.

Reproduction instructions should be precise without exposing unnecessary operational risk. They should describe prerequisites, required permissions, relevant configuration, the test sequence, expected results, actual results, and safe cleanup or restoration steps. If a test could alter flight parameters or disrupt a communication channel, the report should explain the controlled environment and safeguards used. Reproduction should not require a reader to perform hazardous tests on an operational aircraft or interfere with equipment outside the authorized scope.

A finding should distinguish demonstrated facts from interpretation. The observation might establish that an unauthorized configuration change was accepted under specified conditions. The impact analysis may then explain how that change could affect a safety control. If the assessor did not demonstrate the downstream effect, the report should say so. This approach preserves the significance of a credible risk while preventing assumptions from being presented as verified outcomes.

Risk ratings provide a consistent way to prioritize findings. A rating method should consider the ease of exploitation, the access and privileges required, the exposure of the affected interface, the reliability of the observed behavior, the potential consequences, and the effectiveness of existing safeguards. For drones, impact analysis should explicitly consider physical safety and mission integrity in addition to data confidentiality and service availability.

A vulnerability that allows unauthorized alteration of a flight-critical setting may warrant a higher priority than a flaw exposing low-

sensitivity diagnostic information, even if both are technically straightforward to reproduce. Similarly, a weakness requiring prolonged physical access to a protected maintenance interface may have a different likelihood from one exposed through a widely reachable remote service. The assessment should explain these distinctions so that the rating is meaningful to decision-makers.

A scoring framework such as the Common Vulnerability Scoring System can help describe technical severity in a standardized manner when it is appropriate to the finding. However, a generic score should not be treated as a complete assessment of operational risk. A drone's deployment environment, safety constraints, exposure duration, fleet size, and dependence on remote infrastructure may substantially affect priority. Organizations should document how standardized technical severity relates to their own operational risk model.

Confidence should be recorded separately from severity. A high-impact scenario supported only by limited evidence may require urgent validation, but its uncertainty should remain visible. A moderate-impact weakness confirmed across several representative configurations may justify a different response. Separating severity, likelihood, and confidence helps management distinguish between a potentially catastrophic concern that needs immediate investigation and a well-established defect with a narrower impact.

The report should provide an executive-level account for decision-makers and a technical account for the engineers responsible for remediation. The executive section should explain the overall condition of the system, the most significant risks, operational implications, and decisions requiring management attention. The technical section should describe the evidence, affected components, reproduction conditions, root causes, and recommended corrective measures. Both sections should be consistent and should avoid unsupported assurances that the system is either completely secure or fundamentally unsafe.

Remediation guidance should address the underlying cause rather than only the visible symptom. If an administrative service is exposed unnecessarily, disabling the service may be an appropriate immediate mitigation, while architectural changes may be required for a permanent correction. If a command lacks authorization checks, the solution should enforce authorization at the relevant decision point rather than relying solely on network restrictions. If firmware integrity cannot be established, the correction may require changes to the verification chain, signing-key management, update procedures, or recovery design.

Recommendations should be practical and prioritized. Critical findings may require immediate containment or temporary restrictions on use. High-priority findings should receive an assigned owner and a defined correction schedule. Lower-priority findings can be incorporated into planned engineering work, provided the remaining risk is understood and accepted by an authorized decision-maker. Where immediate remediation is impractical, the report should identify compensating controls, their limitations, and the conditions under which they must be reviewed.

Every remediation action should have a means of verification. Installing a patch is not sufficient evidence that the original weakness has been resolved. Retesting should confirm that the problematic behavior no longer occurs under the relevant conditions, that legitimate functions continue to work, and that the correction has not introduced a new weakness. Changes affecting flight-critical behavior should also undergo the required safety and operational validation.

Reports should protect sensitive information. Detailed findings may expose exploitable weaknesses, internal network structures, credentials, aircraft locations, or proprietary implementation details. Distribution should therefore be limited to people with a legitimate need to know. Credentials and cryptographic secrets should never be

included in ordinary reports. Evidence should be retained according to an established policy, with access restrictions and retention periods appropriate to the sensitivity of the material.

A final report is most valuable when it creates a reliable record of the system's security condition at a particular time. It should identify the scope, methods, limitations, findings, risk rationale, and remediation status. It should also distinguish verified vulnerabilities from observations, recommendations, and areas that could not be assessed. This discipline enables future assessors to compare results over time and determine whether the organization is making measurable progress.

## Establishing Responsible Disclosure and Continuous Security Improvement

Drone security research can reveal weaknesses that affect aircraft owners, manufacturers, operators, service providers, and people who have no direct involvement in the system. Responsible disclosure provides a structured way to communicate such findings to the organizations capable of addressing them while reducing the risk of unnecessary exposure. It is not merely a matter of sending a report. It requires sound judgment about authorization, evidence, communication, remediation, and the consequences of publishing technical details.

The process should begin with a clear distinction between authorized assessment and independent discovery. Testing on equipment owned by the researcher may still involve third-party services, shared radio environments, proprietary systems, or networks for which the researcher lacks permission. Before attempting to validate a suspected weakness, the researcher should establish what activities are permitted and avoid actions that could disrupt other users,

interfere with aviation operations, expose private information, or affect aircraft beyond the defined scope.

When a credible vulnerability is identified, the researcher should preserve enough evidence to explain the problem without collecting unrelated sensitive information. The initial report should identify the affected product or component, the relevant version or configuration, the observed behavior, the conditions required to reproduce it, the potential impact, and any practical mitigation already known. The report should distinguish demonstrated results from suspected consequences and should not include unnecessary secrets or data belonging to other parties.

The organization receiving the report needs sufficient information to investigate the issue. Vague claims that a system is insecure are rarely actionable, while a concise, technically accurate explanation can help engineers locate the affected component and assess the risk. Where the evidence is incomplete, the researcher should identify the uncertainty and explain what additional validation may be necessary. Constructive communication helps reduce misunderstanding and encourages a focus on correction rather than disagreement over terminology.

Organizations should establish a security contact and a process for receiving vulnerability reports. A published policy can explain the authorized reporting channel, the information expected from researchers, how reports are acknowledged, how findings are evaluated, and how sensitive evidence is protected. It should also describe the boundaries of permitted testing and clarify that reports will be handled through a defined process. Any commitment concerning response times, confidentiality, or legal treatment should be realistic and consistent with the organization's actual procedures.

Acknowledgment and tracking are important because vulnerability reports can otherwise disappear into ordinary support channels. Each report should receive a reference identifier and an assigned

owner. The organization should record when the report arrived, which component may be affected, the current assessment status, and the decisions made. Reports involving credible risks to flight safety or unauthorized control may require immediate escalation to engineering, security, safety, and operational leadership.

Vulnerability triage should consider technical severity, exposure, exploitability, affected deployments, available mitigations, and potential safety consequences. The organization should determine whether the finding affects one configuration or a wider family of products. It should also examine whether related components share the same implementation or dependency. This broader review helps prevent a correction that addresses one reported device while leaving other affected systems exposed.

Communication should continue throughout remediation. The researcher may be able to clarify reproduction conditions, while the engineering team may need to explain why a particular configuration behaves differently from the one tested. Both parties should preserve a factual record of these exchanges. If the finding cannot be reproduced, the organization should explain what was tested and identify any relevant differences in version, configuration, or environment rather than dismissing the report without examination.

Disclosure timing requires careful judgment. Public release can encourage accountability and help affected users understand risk, but premature publication of detailed exploit instructions may expose systems before a correction is available. The appropriate timing depends on the severity of the issue, the feasibility of mitigation, the extent of deployment, the risk of active exploitation, and any applicable legal or contractual requirements. Researchers and organizations should seek a practical path that supports remediation while avoiding unnecessary harm.

Responsible disclosure does not require every finding to remain confidential indefinitely. In some circumstances, public

communication may be appropriate after a fix or mitigation has been made available, particularly when users need information to make informed security decisions. Any publication should be accurate, proportionate, and careful about details that could facilitate immediate misuse. Where multiple vendors or service providers are involved, coordination may be necessary because one correction may depend on another organization's release or infrastructure changes.

For drone manufacturers and fleet operators, continuous security improvement begins with treating security as an ongoing operational responsibility. A single assessment provides a snapshot, not permanent assurance. Software changes, new payloads, altered network connections, evolving threats, supplier updates, and changes in operating conditions can all affect the system's security. Organizations should therefore maintain a process for reviewing changes, reassessing important risks, and verifying that existing controls continue to work.

A vulnerability management program should connect discovery to remediation and verification. Findings from assessments, incident reports, maintenance reviews, and supplier advisories should enter a common tracking process. Each issue should have a documented severity, owner, status, mitigation, and closure criterion. Overdue high-priority findings should be escalated, and accepted risks should have a named decision-maker and a defined review date. Risk acceptance should not become a permanent substitute for correcting weaknesses that continue to threaten safe operation.

Secure development practices help reduce the number of defects reaching deployed aircraft. Requirements should address authentication, authorization, input validation, software integrity, error handling, logging, and recovery behavior. Code review and automated testing can identify implementation defects early, while dependency analysis can reveal vulnerable third-party components. Security tests should be integrated into development and release workflows, with additional scrutiny applied to changes that affect

communication interfaces, boot processes, cryptographic functions, or flight-critical logic.

Supplier management is equally important. Organizations should understand which components originate from external vendors, how those components are maintained, and whether security updates will remain available throughout the expected service life. Procurement decisions should consider update support, vulnerability reporting practices, software provenance, access to technical documentation, and the ability to recover equipment when a supplier service becomes unavailable. A low purchase price may not represent good long-term value if the platform cannot be maintained securely.

Continuous improvement also requires meaningful performance measures. Counting the number of vulnerabilities discovered provides limited insight because a larger number may reflect either more weaknesses or more effective testing. More useful measures can include the time required to acknowledge and triage reports, the time needed to correct high-priority issues, the proportion of deployed systems receiving verified updates, the number of unresolved risks beyond their review dates, and the success rate of recovery exercises. Measures should encourage genuine risk reduction rather than superficial compliance.

Lessons from incidents and exercises should influence engineering priorities. If a recovery procedure repeatedly fails because configuration backups are outdated, the organization should improve backup validation and restoration testing. If staff frequently bypass authentication because the workflow is impractical, the response should address both the usability problem and the security requirement. If several products share the same vulnerable dependency, development teams should examine how dependencies are selected, tracked, updated, and tested.

Training should evolve alongside the technology. Operators need to understand current reporting procedures and the limits of their

authority. Developers need to understand secure implementation and testing requirements. Maintenance personnel need to know how to verify software, protect credentials, and document configuration changes. Managers need to understand the operational implications of unresolved risks and the resources required to address them. Regular exercises and reviews help confirm that these responsibilities remain clear as personnel and systems change.

Professional practice also requires honesty about assessment limitations. No finite test can establish that a complex drone system contains no vulnerabilities. A responsible assessor explains which components were examined, which methods were used, what conditions were reproduced, and which areas remain uncertain. Recommendations should be based on evidence and proportionate to the risks, while claims about security should remain within the limits of the work performed.

The long-term objective is to create a process in which security findings produce measurable engineering and operational improvements. Architecture reviews inform development decisions; technical assessments reveal weaknesses; incident response tests the effectiveness of safeguards; responsible disclosure brings relevant findings to the right people; and retesting confirms whether corrections work. When these activities are connected through clear accountability and documented evidence, drone security becomes a continuing professional discipline supported by repeatable methods rather than an occasional exercise performed only when a problem becomes visible.

# Frequently Asked Questions About Drone Cybersecurity

## Understanding Drone Cybersecurity

### What is drone cybersecurity?

Drone cybersecurity is the practice of protecting unmanned aircraft, their onboard electronics, control software, communication systems, navigation equipment, ground stations, and connected services against unauthorized access, manipulation, disruption, and data exposure. It combines principles from embedded systems security, network defense, wireless communications, cryptography, software assurance, and aviation safety.

A drone is not simply a flying computer. It is a physical system whose behavior depends on the accuracy and integrity of digital information. If an unauthorized person changes a mission configuration, compromises a control application, or interferes with a critical communication link, the consequences may extend beyond lost data or temporary service interruption. Depending on the aircraft and circumstances, the incident could affect navigation, flight stability, payload operation, or the safety of people and property.

Effective drone cybersecurity therefore seeks to preserve three essential properties: only authorized entities should be able to exercise control, the information used to make operational decisions

should remain trustworthy, and essential functions should remain available or transition to an appropriate safe state when failures occur.

## Why is cybersecurity important for unmanned aircraft?

Unmanned aircraft depend on interconnected electronic and software components that may operate in environments where physical access is limited and communication conditions are unpredictable. Their exposure can extend from onboard hardware and wireless interfaces to maintenance computers, remote management services, and software distribution systems.

A security weakness may allow an attacker to access sensitive mission information, impersonate an authorized device, introduce an unsafe configuration, or disrupt a service needed for normal operation. Even when a weakness does not directly affect flight control, it may provide a starting point for further compromise.

Cybersecurity also supports operational reliability. Authentication, software verification, secure configuration, and monitoring help prevent unauthorized changes while making it easier to identify faults and investigate incidents. These protections are valuable for recreational, commercial, industrial, research, agricultural, and public-service applications.

## Is drone cybersecurity only concerned with hackers taking control of aircraft?

No. Unauthorized control is only one possible outcome. Drone cybersecurity also addresses the theft of mission data, exposure of video recordings, disclosure of operational locations, compromise of operator accounts, malicious firmware changes, unauthorized access to maintenance interfaces, and disruption of supporting services.

A drone may remain controllable while still experiencing a serious security incident. For example, sensitive recordings could be copied from an inadequately protected storage device without affecting flight. Similarly, a compromised ground station might expose credentials that could be misused later.

A comprehensive security assessment examines confidentiality, integrity, availability, and physical safety. This broader perspective prevents organizations from overlooking important risks simply because an aircraft has not exhibited obvious control problems.

## Which parts of a drone require cybersecurity protection?

Protection may be required for the flight controller, companion computer, firmware, navigation sensors, radio equipment, ground control station, payload electronics, local networks, remote services, credentials, mission files, and maintenance tools.

The precise requirements depend on the aircraft's architecture. A basic aircraft may have relatively few connected components, whereas an advanced platform may combine multiple communication links, onboard applications, cameras, cellular connectivity, and cloud-based management.

The supporting equipment deserves particular attention. A secure aircraft can still be exposed through a compromised operator computer or an update process that distributes unverified software. Security planning should therefore include every component that can influence aircraft behavior, handle sensitive information, or establish trust with another system.

## What is the difference between drone safety and drone cybersecurity?

Drone safety concerns the prevention of accidents and unacceptable harm during operation. Drone cybersecurity focuses on protecting systems, information, and functions against unauthorized access, manipulation, and disruption. The two disciplines overlap because a cybersecurity failure can create a physical hazard.

A navigation sensor failure, for example, may result from an ordinary hardware defect, environmental interference, or malicious manipulation. Safety engineering examines the potential consequences and required safeguards, while cybersecurity investigates whether an intentional or exploitable weakness contributed to the condition.

A mature engineering program treats security and safety as related responsibilities. Security controls should not introduce unsafe failure modes, and safety mechanisms should not depend entirely on the assumption that every digital input is trustworthy.

# Flight Controllers, Firmware, and Embedded Systems

## What does a flight controller do, and why must it be protected?

The flight controller processes sensor measurements and control instructions to manage the aircraft's movement. Depending on the design, it may support stabilization, attitude control, navigation, motor commands, flight modes, and emergency responses.

Because it influences safety-critical behavior, its software and configuration require strong integrity protections. Unauthorized modifications to control logic or critical parameters could change how the aircraft responds to commands, sensor information, or failures.

Appropriate safeguards include verified firmware, controlled configuration changes, restricted diagnostic interfaces, secure communication with other components, and independent checks on safety-sensitive commands. Testing should establish how these protections behave under normal conditions and when verification fails.

## What is drone firmware?

Firmware is software closely associated with electronic hardware. In a drone, it may control the flight controller, communication module,

battery-management electronics, sensors, or other embedded devices.

Firmware determines how hardware functions and how it responds to external inputs. Its security depends on design quality, correct implementation, secure installation, and ongoing maintenance.

A firmware release should be identifiable, compatible with the intended hardware, and obtained from a trusted source. Where supported, cryptographic signatures and secure boot can help verify authenticity and integrity. These mechanisms do not guarantee that the firmware contains no defects, so software review and testing remain necessary.

## What is secure boot?

Secure boot is a mechanism that verifies software against defined trust requirements during startup before allowing it to execute. A typical design begins with a protected verification component that checks the next stage of the boot process.

The security of this arrangement depends on the protection of the initial trust mechanism, the verification rules, and the keys used to establish authenticity. If unauthorized software can bypass verification or if signing credentials are compromised, the protection may be weakened.

Secure boot should be evaluated alongside update verification, recovery procedures, and version controls. It is one part of software integrity, not a complete cybersecurity solution by itself.

## Why are firmware updates important for drone security?

Firmware updates can correct vulnerabilities, improve reliability, address compatibility problems, and strengthen security controls. An aircraft that remains on an outdated release may retain weaknesses that have already been corrected by its manufacturer or development team.

However, an update should not be installed without considering compatibility and operational risk. A new release may change system behavior or introduce defects. Organizations should review release information, verify package authenticity, test important changes, maintain suitable recovery options, and confirm that the aircraft operates correctly after installation.

Updates affecting flight-critical components should follow the applicable maintenance and validation procedures. The goal is to keep the system secure without compromising its operational safety.

## What is the difference between a firmware vulnerability and a firmware defect?

A defect is an error or unexpected behavior in software. A vulnerability is a weakness that can be exploited or otherwise lead to a security failure under relevant conditions.

Some defects have no meaningful security impact. Others may permit unauthorized access, bypass authentication, expose sensitive information, or alter safety-critical behavior. A defect becomes a

security concern when its effects threaten a protected asset or violate an important security requirement.

Assessors should document the conditions under which the issue occurs, the privileges required, the affected versions, and the potential consequences. This evidence supports accurate prioritization and avoids overstating the significance of ordinary software faults.

## What are debug ports, and why can they create security risks?

Debug ports are interfaces used to inspect, test, or troubleshoot electronic systems during development and maintenance. Depending on the hardware, they may provide access to memory, processor state, diagnostic output, or programming functions.

If these interfaces remain accessible in a production device without adequate restrictions, they may expose sensitive information or permit unauthorized modification. The risk depends on the capabilities of the interface, physical access requirements, hardware protections, and the device's security architecture.

Organizations should identify diagnostic interfaces, document their purpose, restrict access, and disable or protect functions that are unnecessary during normal operation. Maintenance access should be controlled and restored to the approved configuration when work is complete.

# Wireless Communications and Drone Protocols

## Can a drone's wireless communication be compromised?

Wireless communications can be exposed to several risks, including unauthorized access, interception, interference, impersonation, and manipulation of messages. The actual risk depends on the communication technology, security configuration, authentication mechanisms, environmental conditions, and capabilities of a potential attacker.

A wireless link should not be considered secure simply because it has limited range or uses a dedicated frequency. Effective protection requires appropriate device authentication, message integrity, authorization, and encryption where needed.

Testing should take place within an authorized environment. Uncontrolled experiments can interfere with unrelated equipment and may create safety or legal problems, especially when aircraft are operating nearby.

## Does encryption prevent every form of wireless attack?

No. Encryption helps protect information from unauthorized disclosure, but it does not automatically prevent every security

failure. A poorly implemented system may accept commands from an authenticated but compromised account, fail to validate message content, or permit unauthorized actions through another interface.

Encryption also depends on sound key management, correct protocol implementation, and appropriate endpoint verification. If cryptographic keys are exposed, the intended protection may be undermined.

A complete communication-security design combines encryption with authentication, authorization, message-integrity protection, sensible session management, and safeguards against replayed or invalid commands.

## What is MAVLink, and why is it relevant to drone security?

MAVLink is a lightweight messaging protocol commonly used in unmanned vehicle systems to exchange telemetry, commands, and status information between components such as aircraft controllers, companion computers, and ground stations.

The protocol defines message structures and communication conventions, but secure operation also depends on the surrounding architecture and implementation. Assessors must determine how identities are verified, which commands are authorized, how untrusted messages are handled, and whether security-sensitive operations receive appropriate protection.

The presence of a protocol does not automatically establish that a system is secure or insecure. Its security must be evaluated in the

context of the particular implementation, configuration, and operational requirements.

## What is the difference between authentication and authorization?

Authentication verifies identity. Authorization determines which actions that identity is permitted to perform.

For example, an operator may successfully authenticate to a ground station but have permission only to monitor telemetry rather than modify firmware. A maintenance technician may be authorized to inspect diagnostic information without being permitted to initiate a mission.

Separating these functions helps prevent a valid account from receiving excessive authority. Sensitive actions should be limited to appropriate roles, and systems should enforce permissions at the point where the action is accepted rather than relying solely on the user's claimed identity.

## What is a replay attack?

A replay attack occurs when an attacker captures a previously valid message or exchange and sends it again in an attempt to trigger an unauthorized or unintended action.

Whether replay is possible depends on the protocol and implementation. Defenses may include sequence numbers,

timestamps, nonces, authenticated session state, or other mechanisms that help a receiver recognize stale or duplicated messages.

These protections must be implemented correctly and tested against the system's actual communication behavior. Encryption alone does not necessarily prevent replay if the protocol does not distinguish a fresh message from a previously valid one.

### What is the difference between jamming and spoofing?

Jamming involves interference intended to degrade or prevent reception of a signal. Spoofing involves presenting deceptive signals or data intended to be accepted as legitimate.

Both can affect navigation or communication, but their mechanisms and observable effects differ. A receiver experiencing jamming may lose a signal or report reduced quality. A receiver exposed to spoofing may continue producing data that appears plausible but is incorrect.

Detection and response depend on the system. Appropriate protections may include monitoring signal quality, comparing independent sensors, checking data consistency, and following validated contingency procedures. Neither condition should be diagnosed solely from a single unusual measurement.

## Navigation and Position Integrity

## How does a drone determine its position?

Many drones use a global navigation satellite system to estimate their position and timing. Depending on the aircraft, this information may be combined with inertial measurements, barometric pressure, magnetic heading, optical sensors, cameras, or other data sources.

The navigation system processes these measurements to estimate the aircraft's state. Each source has limitations. Satellite signals may be unavailable or degraded, inertial estimates can accumulate error, and individual sensors may produce inaccurate measurements under certain conditions.

Combining independent sources can improve resilience, but only when the system evaluates their quality and handles disagreements appropriately. Sensor fusion should not blindly accept every input.

## Can a drone continue operating if satellite navigation becomes unavailable?

Some aircraft can maintain selected functions using inertial navigation, optical flow, visual navigation, or other onboard capabilities. The available options depend on the hardware, software, environment, and operating mode.

Loss of satellite navigation can increase uncertainty in position and movement estimates. Inertial systems may accumulate drift, while optical methods can perform poorly in conditions with limited texture, poor lighting, or insufficient visual information.

The aircraft's response should follow its validated design and approved operating procedures. Operators should understand which functions remain available, what limitations apply, and which recovery actions are appropriate when navigation confidence falls below an acceptable level.

## What is navigation integrity?

Navigation integrity concerns whether positioning and navigation information can be trusted for its intended purpose and whether the system can identify when that information may be unreliable.

Accuracy and integrity are related but different. Accuracy describes how close an estimate is to the actual value. Integrity includes the ability to recognize unacceptable errors and communicate when a navigation solution should not be relied upon.

A system may produce an apparently precise position estimate while using corrupted information. Monitoring should therefore consider consistency across sensors, signal quality, expected movement, and the confidence associated with the navigation solution.

## How can navigation anomalies be detected?

Navigation anomalies may be identified by comparing independent sensor measurements, examining sudden changes in estimated position, checking the consistency of movement and heading, and monitoring the quality of satellite signals.

No single indicator is sufficient in every circumstance. A sharp position change may result from poor reception, a sensor fault, an environmental condition, or a deliberate attempt to deceive the receiver.

A robust system combines several checks and applies a defined response when confidence becomes inadequate. Testing should use simulation or other controlled methods to verify that the aircraft identifies relevant anomalies and transitions safely.

# Companion Computers, Networks, and Payloads

## What is a companion computer, and how does it affect security?

A companion computer is an onboard computing platform that performs tasks alongside the flight controller. It may process imagery, run higher-level autonomy software, manage payloads, communicate with remote services, or analyze mission data.

Because it may run a general-purpose operating system and additional applications, it can introduce services and interfaces that require protection. A compromised companion computer should not automatically gain unrestricted authority over flight-critical functions.

Security measures include application isolation, restricted permissions, secure updates, network segmentation, protected

credentials, and narrowly defined interfaces with the flight controller. The architecture should make clear which functions the companion computer can influence and which remain independently protected.

## Why is the ground control station a security concern?

The ground control station may store credentials, mission files, aircraft configurations, telemetry, and other sensitive information. It may also have permission to issue commands or manage multiple aircraft.

If the station is compromised, an attacker may be able to misuse its legitimate access. Risks can be reduced through timely software updates, restricted administrative privileges, strong authentication, protected storage, endpoint security, and controlled access to the networks used for flight operations.

Organizations should also consider the station's physical security and maintenance practices. An unprotected laptop used to configure several aircraft can become a common point of exposure across an entire fleet.

## What is network segmentation, and why does it matter?

Network segmentation separates a network into zones with controlled communication between them. It can limit the extent to which a compromised component can reach other systems.

For example, payload-processing software may need access to camera data but not to administrative services on the ground station. A remote telemetry service may need selected status information without requiring direct access to all flight-control functions.

Segmentation is effective only when its rules are correctly configured and enforced. Assessors should verify actual communication paths, permitted services, and access restrictions rather than relying solely on architectural diagrams.

## What information should be protected on a drone?

Sensitive information may include operator credentials, cryptographic keys, mission plans, flight logs, video recordings, location history, network settings, software packages, and configuration files.

The level of protection should reflect the sensitivity of the information and the consequences of disclosure or modification. For example, unauthorized disclosure of location records may reveal operational patterns, while modification of a mission file could affect where the aircraft attempts to fly.

Appropriate safeguards include access control, encryption where suitable, protected storage, secure data transfer, retention limits, and controlled disposal of equipment.

# Security Testing and Vulnerability Assessment

## What is a drone cybersecurity assessment?

A drone cybersecurity assessment is a structured examination of an aircraft and its supporting systems to identify weaknesses, evaluate their consequences, and recommend practical improvements.

The assessment may include architecture review, firmware analysis, configuration inspection, protocol testing, network exposure review, authentication checks, and evaluation of update and recovery mechanisms. The scope depends on the system and the purpose of the engagement.

A professional assessment begins with authorization and documented boundaries. Testing should use suitable environments, protect operational safety, and produce evidence that supports each finding.

## What is the difference between vulnerability assessment and penetration testing?

A vulnerability assessment focuses on identifying, analyzing, and prioritizing weaknesses. It may rely on configuration reviews, software analysis, automated tools, documentation, and controlled tests.

Penetration testing attempts to determine whether selected weaknesses can be used to reach defined objectives under authorized conditions. It generally emphasizes demonstrating practical impact rather than simply identifying a possible defect.

The methods can complement one another. Neither guarantees that every weakness has been discovered, and both should state their scope, limitations, and testing conditions.

## What is fuzz testing?

Fuzz testing evaluates software by supplying a variety of unexpected, malformed, or unusual inputs and observing how the system responds. It can reveal crashes, memory errors, inconsistent behavior, and weaknesses in input handling.

For drone systems, fuzz testing may be useful for communication parsers, configuration readers, data-processing applications, or other components that receive structured input. Testing should occur in an isolated environment whenever a failure could affect flight safety or operational equipment.

A crash does not automatically establish a security vulnerability. The assessor must determine whether the failure can affect security properties, whether it is reproducible, and what conditions are required for the behavior to occur.

## Why is written authorization important?

Authorization defines which systems may be tested and which actions are permitted. It protects the organization and assessor by establishing clear boundaries and reducing the chance of accidental disruption or unauthorized access.

A written agreement should identify the assets, permitted methods, test window, contacts, prohibited activities, data-handling requirements, and emergency stopping conditions. It should also clarify whether connected services or third-party systems are included.

Testing should not extend beyond those boundaries merely because a related system becomes technically reachable. Permission to assess one aircraft does not automatically authorize interference with other aircraft, public communication infrastructure, or external services.

## How should a security finding be documented?

A useful finding identifies the affected component, the observed weakness, the conditions under which it occurs, the potential impact, and the evidence supporting the conclusion. It should also provide a practical recommendation for reducing the risk.

The report should distinguish confirmed behavior from assumptions. If a test establishes that a particular permission is missing, the finding can state that directly. If a wider operational consequence is plausible but was not demonstrated, the report should explain the reasoning and the uncertainty.

Clear documentation helps engineers reproduce the result, prioritize corrective work, and verify that the weakness has been addressed.

### How are drone cybersecurity risks prioritized?

Risk prioritization considers the likelihood of a weakness being exploited or causing a failure, the impact of the resulting event, the access required, the exposure of the affected component, and the effectiveness of existing safeguards.

For drones, physical safety and mission integrity deserve explicit consideration. A weakness affecting flight-critical control may warrant greater urgency than one exposing low-sensitivity diagnostic information, even if both are technically easy to reproduce.

A standardized severity framework can support consistent evaluation, but the final priority should also reflect the aircraft's operating environment, deployment scale, mission requirements, and applicable safety obligations.

## Incident Response and Maintenance

### What should an operator do if a drone is suspected of being compromised?

The immediate priority is the safety of people and property. The operator should follow the aircraft's approved emergency and contingency procedures, avoid untested interventions, and seek to place the aircraft in an appropriate safe state.

Once immediate hazards are controlled, the organization should document the observed behavior, identify potentially affected systems, preserve available logs, and notify the responsible technical or security personnel. The aircraft should not automatically return to service simply because the visible symptom has disappeared.

Recovery should include a documented evaluation of software integrity, configuration, credentials, and relevant hardware. Required testing should confirm that the system is trustworthy and safe for its intended operation.

## What is incident response?

Incident response is the organized process of preparing for, detecting, analyzing, containing, resolving, and learning from security incidents.

For a drone operation, it may involve flight operators, maintenance personnel, software engineers, security specialists, and management. Their responsibilities should be defined before an incident occurs.

A sound process protects people first, limits further compromise, preserves evidence when practicable, restores a trustworthy operating state, and identifies improvements that reduce the chance of recurrence.

## How often should drone firmware and software be updated?

There is no single interval suitable for every drone. Updates should be guided by the availability and importance of security fixes, the aircraft's exposure, the vendor's support policy, compatibility requirements, and the risks associated with changing flight-critical software.

Organizations should monitor relevant release information, assess whether an update applies to their equipment, and test important changes before deployment where practicable. Urgent vulnerabilities may require accelerated action, but the process should still include validation and a documented decision.

An update schedule should also identify unsupported components and establish how unresolved vulnerabilities will be managed.

## Why are logs important during an investigation?

Logs record events that help investigators reconstruct system behavior. Depending on the platform, they may contain authentication attempts, mission changes, software updates, configuration modifications, communication events, flight-mode transitions, and error messages.

Reliable logs can help establish what happened, when it happened, and which components were involved. Their usefulness depends on completeness, accurate timing, protection against unauthorized modification, and appropriate retention.

Organizations should define which events must be recorded, who may access the records, and how evidence will be preserved during an incident.

### What is responsible vulnerability disclosure?

Responsible vulnerability disclosure is the process of communicating a security weakness to the relevant organization so it can be evaluated and addressed while unnecessary harm is minimized.

A good report describes the affected component, the observed behavior, the conditions required to reproduce it, and the potential consequences. Researchers should avoid unauthorized testing, unnecessary collection of sensitive data, and premature publication of details that could expose users to avoidable risk.

Organizations should maintain a clear reporting channel, acknowledge reports, assess their severity, coordinate remediation, and communicate responsibly with affected stakeholders.

## Professional Practice and Long-Term Security

### Can a drone ever be completely secure?

No complex connected system can be guaranteed to be completely secure under every possible condition. Hardware, software, people, communication infrastructure, and operating environments introduce changing risks.

The practical objective is to reduce risk to an acceptable level, identify weaknesses before they cause harm, detect abnormal behavior, and maintain reliable recovery procedures. Security assurance depends on the evidence available, the scope of testing, and the conditions under which the system is expected to operate.

Claims about security should therefore be specific. A test may establish that an update mechanism rejects unauthorized packages under defined conditions, but it cannot establish that the entire aircraft is free of vulnerabilities.

## What skills are useful for a drone cybersecurity professional?

A professional benefits from knowledge of embedded systems, operating systems, networking, wireless communications, cryptography, software analysis, and basic aviation safety principles. Familiarity with firmware architecture, authentication mechanisms, secure configuration, testing methods, and incident response is also valuable.

Communication skills are important because technical findings must be explained to engineers, operators, managers, and other stakeholders. A good practitioner can describe the evidence, explain the operational consequences, acknowledge uncertainty, and recommend realistic corrective measures.

Ethical judgment and respect for authorization boundaries are essential. Technical capability must be matched by disciplined testing practices and an understanding of the possible consequences of experimentation.

## What is the role of documentation in drone security?

Documentation establishes how a system is intended to operate, which configurations are approved, how software is maintained, and who is responsible for security decisions. It supports troubleshooting, incident response, regulatory review, and the transfer of knowledge between teams.

Important records may include architecture diagrams, asset inventories, approved configurations, update histories, access-control policies, assessment reports, incident records, and recovery procedures.

Documentation must be maintained as the system changes. Outdated diagrams or incorrect firmware inventories can create a false sense of assurance and lead assessors to overlook actual exposure.

## How can organizations maintain security across a fleet of drones?

Fleet security begins with a reliable inventory of aircraft, hardware revisions, software versions, communication modules, credentials, and assigned operators. Without this information, it is difficult to determine which aircraft are affected by a vulnerability or whether a security update has reached every relevant device.

Organizations should use consistent configuration standards, controlled maintenance procedures, credential lifecycle management, monitoring, and periodic assessments. Shared services

and maintenance computers deserve special attention because a single compromise may affect several aircraft.

Fleet-wide reviews should be triggered by significant vulnerabilities, supplier advisories, major software changes, and relevant incidents. Retesting should confirm that corrective measures are effective across the configurations to which they apply.

## What should be considered when purchasing a drone from a cybersecurity perspective?

Purchasers should consider the availability of security updates, the supplier's vulnerability-reporting process, the integrity of firmware installation, authentication and access-control capabilities, data storage and privacy, and the ability to maintain the aircraft throughout its expected service life.

It is also important to understand which components are supplied by third parties, whether remote services are required for essential functions, and what happens if a supplier discontinues support. Technical documentation and clear maintenance procedures can make a substantial difference to long-term security.

A purchase decision should account for the total cost of secure operation, including updates, staff training, maintenance, and eventual retirement of the equipment. A system that is inexpensive initially may become difficult to protect if its software is no longer maintained.

## How does continuous security improvement work in practice?

Continuous improvement links assessment, maintenance, incident response, and engineering changes into a recurring process. Findings are recorded, assigned to responsible owners, prioritized, corrected, and retested. Lessons from incidents and exercises inform updates to technical controls and operating procedures.

Organizations can measure progress through indicators such as the time required to address high-priority findings, the proportion of supported aircraft receiving verified updates, the number of overdue remediation actions, and the success of recovery exercises.

These measures should reflect meaningful risk reduction rather than the number of reports produced. The most useful security program is one that identifies important weaknesses, corrects them reliably, verifies the results, and adapts as the aircraft and its operating environment change.

# Glossary of Drone Cybersecurity Terms

This glossary defines the principal technical, operational, and security terms used in drone cybersecurity. It covers aircraft architecture, embedded computing, firmware integrity, wireless communications, navigation, cryptography, network security, vulnerability assessment, incident response, and professional security practice. The definitions are written to support readers who are developing their technical knowledge as well as experienced practitioners who need a consistent reference when reviewing drone systems.

# Aircraft Architecture and Flight Control

| Term | Definition |
| --- | --- |
| Actuator | A component that converts a control signal into physical movement. In a drone, actuators may operate motors, control surfaces, landing mechanisms, or payload equipment. |
| Autonomous Flight | Flight in which an aircraft performs some or all navigation and control tasks according to programmed instructions and onboard decision-making, with the level of human supervision determined by the system and operation. |
| Autopilot | An onboard control system that manages flight functions such as stabilization, navigation, altitude regulation, and execution of approved flight commands. |
| Avionics | The electronic systems used for aircraft operation, including flight control, navigation, |

| | |
|---|---|
| | communications, monitoring, and related functions. |
| Command Authority | The recognized permission to issue instructions that influence an aircraft or one of its subsystems. |
| Companion Computer | An onboard computing platform that performs tasks such as image processing, mission support, data analysis, or higher-level autonomy, often alongside a separate flight controller. |
| Control Loop | A repeated process in which a controller reads sensor information, calculates a response, and adjusts system behavior to achieve a desired state. |
| Electronic Speed Controller (ESC) | An electronic device that regulates the electrical power supplied to a motor, allowing the flight-control system to manage motor speed. |
| Fail-Safe | A predefined response intended to reduce danger when a system detects a fault or loses an essential capability. |

| | |
|---|---|
| Flight Controller | The onboard computer responsible for executing essential flight-control functions and processing information from sensors, operator commands, and other authorized sources. |
| Flight Envelope | The range of operating conditions within which an aircraft is designed or approved to operate safely, including relevant limits on speed, altitude, attitude, and other parameters. |
| Flight Mode | A defined operating state that determines how the aircraft interprets commands and manages flight, such as manual control, assisted flight, or autonomous mission execution. |
| Geofence | A virtual geographic boundary used to define an allowed, restricted, or otherwise significant operating area. Its effectiveness depends on accurate positioning, configuration integrity, and enforcement by the aircraft or supporting system. |
| Ground Control Station (GCS) | A computer, application, or |

| | |
|---|---|
| | integrated system used by an operator to configure an aircraft, plan missions, issue commands, and monitor telemetry. |
| Hardware-in-the-Loop (HIL) | A testing approach that connects real hardware to a simulated environment so that engineers can evaluate system behavior under controlled conditions. |
| Mission Plan | A set of instructions defining an intended flight, potentially including waypoints, altitudes, actions, operating limits, and other mission parameters. |
| Multirotor | An aircraft that uses multiple powered rotors to generate lift and control movement. |
| Payload | Equipment carried by a drone to perform a task, such as a camera, sensor, communication device, or specialized instrument. |
| Redundancy | The provision of additional components, capabilities, or paths so that a single failure does not necessarily cause the loss of an essential function. |

| | |
|---|---|
| Return-to-Home (RTH) | A predefined recovery function that directs an aircraft toward a designated home or recovery location under specified conditions. Its safety depends on configuration, navigation quality, remaining energy, and environmental circumstances. |
| Safety-Critical Function | A function whose failure or compromise could create a significant hazard to people, property, or the aircraft. |
| Telemetry | Data describing the aircraft's status and operating conditions, including information such as position, altitude, battery state, communication quality, and system health. |
| Waypoint | A defined geographic or navigational target used to guide a planned flight route. |

# Embedded Hardware, Firmware, and Software

| Term | Definition |
| --- | --- |
| Application Programming Interface (API) | A defined interface through which software components exchange information or request functions from one another. |
| Attack Surface | The collection of interfaces, services, components, and processes through which a system could potentially be accessed or affected by an unauthorized party. |
| Bootloader | Software that executes during startup and prepares the system to load or execute the main firmware or operating system. |
| Central Processing Unit (CPU) | The processor that executes general-purpose instructions within a computing system. |
| Configuration Baseline | A documented set of approved settings and component versions |

| | |
|---|---|
| | used as a reference for detecting unauthorized or unexpected changes. |
| Debug Interface | A hardware or software interface used to inspect, test, or troubleshoot a system during development and maintenance. |
| Dependency | A software component, library, service, or other resource required for another component to operate correctly. |
| Embedded System | A computing system designed to perform specific functions within a larger device, such as a flight controller, sensor module, or onboard communication unit. |
| Firmware | Software stored in or closely associated with hardware that controls the device's behavior and interaction with electronic components. |
| Firmware Analysis | The examination of firmware to understand its structure, functionality, security properties, and possible weaknesses. |

| | |
|---|---|
| Firmware Image | A packaged representation of firmware intended for installation, storage, verification, or execution on a compatible device. |
| Hardware Abstraction Layer (HAL) | Software that provides a consistent interface to hardware details, reducing the need for higher-level applications to interact directly with individual devices. |
| Input Validation | The process of checking received data against expected formats, ranges, types, and rules before the system uses it. |
| Interprocess Communication (IPC) | Mechanisms that allow separate software processes to exchange data or coordinate actions. |
| Library | Reusable software code that provides functions or services to other applications. |
| Microcontroller Unit (MCU) | An integrated circuit containing a processor, memory, and peripheral interfaces for controlling electronic devices. |

| | |
|---|---|
| Memory Protection | Hardware or software mechanisms that restrict which components can access particular areas of memory. |
| Operating System (OS) | Software that manages a computer's resources and provides services for applications. |
| Privilege Escalation | A condition in which a user or process obtains permissions beyond those originally assigned. |
| Read-Only Memory (ROM) | Memory designed primarily for storing information that is not ordinarily changed during normal operation. In modern systems, the term can refer to different implementations with varying update capabilities. |
| Reverse Engineering | The systematic analysis of a system or software to understand its design, behavior, or implementation. |
| Secure Configuration | A set of settings designed to reduce unnecessary exposure, |

| | |
|---|---|
| | enforce required protections, and restrict access to system functions. |
| Software Bill of Materials (SBOM) | A structured inventory of software components and dependencies included in a product or software release. |
| Source Code | Human-readable program instructions from which software is built or interpreted. |
| Static Analysis | Examination of source code, binaries, or other artifacts without executing the target program. |
| Update Mechanism | The process and supporting components used to deliver, verify, install, and manage software or firmware updates. |

# Drone Communications and Wireless Security

| Term | Definition |
| --- | --- |
| Authentication | The process of verifying the identity of a user, device, service, or other entity. |
| Availability | The property of a system or resource being accessible and usable when required by an authorized party. |
| Bandwidth | The capacity of a communication channel to carry information over a period of time, often expressed in bits per second. |
| Bluetooth | A family of short-range wireless communication technologies used to exchange data between compatible devices. |
| Cellular Connectivity | Communication through a mobile network infrastructure |

| | |
|---|---|
| | that provides coverage across multiple geographic areas. |
| Channel Interference | The degradation of communication caused by unwanted signals or energy affecting a receiver or communication channel. |
| Command Link | A communication path used to transmit instructions to an aircraft or one of its control systems. |
| Communication Link | A wired or wireless path through which devices exchange information. |
| Confidentiality | The property that information is accessible only to authorized people, systems, or processes. |
| Encryption | The transformation of readable information into a protected form using a cryptographic method and key, intended to prevent unauthorized parties from understanding the content. |
| Frequency Band | A specified range of radio |

| | |
|---|---|
| | frequencies used for communication or other radio applications. |
| Frequency-Hopping Spread Spectrum (FHSS) | A radio technique in which communication changes among multiple frequencies according to an agreed pattern. Its security and interference resistance depend on implementation and do not replace authentication or encryption. |
| Gateway | A system that connects different networks or communication environments and may route, filter, or translate traffic between them. |
| Jamming | The deliberate transmission or generation of interference intended to degrade or prevent communication or signal reception. |
| Latency | The time required for information to travel through a system or for a system to respond to an input. |
| Link Budget | An engineering calculation of the gains and losses affecting a |

| | |
|---|---|
| | communication signal between transmitter and receiver. |
| Man-in-the-Middle Attack | An attack in which an unauthorized party positions itself between communicating entities to observe, relay, or potentially alter their exchange. |
| Message Authentication Code (MAC) | A cryptographic value used to help verify a message's integrity and authenticity when the communicating parties share an appropriate secret key. |
| Packet | A formatted unit of data transmitted over a network. |
| Protocol | A set of rules defining how systems format, transmit, receive, and interpret information. |
| Radio Frequency (RF) | The portion of the electromagnetic spectrum commonly used for radio communication and related applications. |
| Replay Attack | An attack in which a previously |

| | |
|---|---|
| | valid message or exchange is captured and transmitted again in an attempt to cause an unauthorized action. |
| Satellite Communication | The transmission of information through a communication system that uses satellites as part of the signal path. |
| Signal-to-Noise Ratio (SNR) | A measure comparing the strength of a desired signal with the strength of background noise or unwanted signal energy. |
| Spoofing | The presentation of false or misleading identity, signal, message, or data intended to be accepted as legitimate. |
| Transport Security | Protection applied to information exchanged between communicating endpoints, commonly including encryption and authentication. |
| Wireless Access Point | A device that allows compatible wireless devices to connect to a network. |

| | |
|---|---|
| Wireless Fidelity (Wi-Fi) | A widely used family of wireless local-area networking technologies. |
| Wireless Reconnaissance | Authorized observation and documentation of wireless devices, channels, configurations, and communication behavior to understand a system's exposure. |

# Drone Protocols, Networking, and Remote Connectivity

| Term | Definition |
| --- | --- |
| Access Control | Rules and mechanisms that determine which identities may access particular resources or perform specified actions. |
| Application Isolation | Separation of software applications or processes to restrict their ability to interfere with one another or access protected resources. |
| Authentication Token | A value used to represent or prove an authenticated session or grant access under a defined security scheme. |
| Cellular Gateway | A device or service that provides a connection between a cellular network and another network or system. |
| Client-Server Architecture | A computing arrangement in which a client requests data or |

| | |
|---|---|
| | services from a server. |
| Distributed System | A system composed of multiple connected computers or devices that coordinate to provide a service or complete a task. |
| Firewall | A security control that permits or blocks network traffic according to defined rules. |
| Internet Protocol (IP) | A networking protocol that supports the addressing and routing of data packets between connected networks. |
| Least Privilege | The principle that users, applications, and devices should receive only the permissions necessary for their assigned tasks. |
| MAVLink | A lightweight messaging protocol commonly used in unmanned vehicle systems to exchange telemetry, commands, and status information. Its security depends on the surrounding architecture and implementation rather than the message format alone. |

| | |
|---|---|
| Network Exposure | The extent to which a system's services, interfaces, or resources are reachable from particular networks or devices. |
| Network Segmentation | The division of a network into separate zones to restrict communication and limit the spread of compromise. |
| Port | A logical communication endpoint used by networked software to distinguish services or applications. |
| Remote Access | The ability to connect to and use a system from a separate physical location or network. |
| Remote Procedure Call (RPC) | A mechanism that allows one software component to request that a function be executed by another component. |
| Session | A period of interaction between systems or users, often associated with authentication and maintained state. |

| | |
|---|---|
| Service | A software function made available to other applications, devices, or users through a defined interface. |
| Socket | A software endpoint used for communication between processes, commonly across a network. |
| Transmission Control Protocol (TCP) | A transport protocol that provides an ordered, reliable byte stream between communicating endpoints. |
| User Datagram Protocol (UDP) | A transport protocol that sends datagrams without providing TCP's built-in guarantees of delivery, ordering, or retransmission. |
| Virtual Private Network (VPN) | A technology that creates a protected logical connection over another network, commonly using encryption and authentication. |
| Vulnerability | A weakness in design, implementation, configuration, or operation that could be exploited or otherwise cause a security failure. |

| | |
|---|---|
| Zero Trust | A security approach that avoids automatically trusting users or devices solely because of their network location and instead requires appropriate verification and access controls. |

# Navigation, Positioning, and Sensor Integrity

| Term | Definition |
| --- | --- |
| Dead Reckoning | Estimating a current position from a previously known position and measurements of movement over time. Errors can accumulate as the estimate continues. |
| Global Navigation Satellite System (GNSS) | A satellite-based system that provides positioning, navigation, and timing information. |
| Global Positioning System (GPS) | A specific satellite-based navigation system operated by the United States. It is one of several systems that may contribute to GNSS-based positioning. |
| Inertial Measurement Unit (IMU) | A sensor assembly that typically measures acceleration and angular velocity and may include additional sensing capabilities. |

| | |
|---|---|
| Inertial Navigation System (INS) | A navigation system that estimates movement and position using inertial sensor measurements, often combined with other information sources. |
| Magnetometer | A sensor that measures magnetic fields and may be used to estimate heading. |
| Navigation Integrity | The degree of confidence that navigation information is sufficiently trustworthy for its intended use, including the ability to identify when it should not be relied upon. |
| Position Estimate | A calculated representation of an object's location based on sensor measurements, navigation algorithms, or external information. |
| Sensor Fusion | The process of combining measurements from multiple sensors or information sources to produce a more useful estimate of system state. |
| Sensor Drift | A gradual change in a sensor's output or estimated state that |

| | |
|---|---|
| | causes increasing error over time. |
| Signal Spoofing | The creation or transmission of deceptive signals intended to cause a receiver to calculate or accept incorrect information. |
| Timing Source | A source of time information used to synchronize devices, timestamp events, or support navigation and communication functions. |
| Trusted Sensor Input | Sensor data that has been evaluated against appropriate integrity, plausibility, and source requirements before being relied upon for a decision. |

# Cryptography, Identity, and Software Integrity

| Term | Definition |
| --- | --- |
| Asymmetric Cryptography | Cryptography that uses a mathematically related public key and private key for operations such as digital signatures or key establishment. |
| Certificate | A digitally signed data structure that binds an identity or other attributes to a public key under a defined trust system. |
| Certificate Authority (CA) | An entity responsible for issuing or signing certificates within a public-key infrastructure. |
| Cryptographic Hash | A fixed-length value calculated from data using a hash function, commonly used for integrity checking and identification. |
| Cryptographic Key | Information used by a cryptographic algorithm to perform operations such as |

| | |
|---|---|
| | encryption, decryption, signing, or verification. |
| Digital Signature | A cryptographic mechanism that enables verification of a message or software package's integrity and its association with the holder of a corresponding private key. |
| Key Rotation | The planned replacement of cryptographic keys to reduce prolonged reliance on a single key and limit the consequences of exposure. |
| Key Revocation | The process of withdrawing trust in a key or credential that should no longer be accepted. |
| Mutual Authentication | An authentication process in which both communicating parties verify the identity of the other. |
| Nonrepudiation | A security objective intended to provide evidence supporting the attribution of an action or message to a particular entity, subject to the strength of the implementation and the surrounding procedures. |

| | |
|---|---|
| One-Time Password (OTP) | A password or code intended for a single use or a limited validity period. |
| Public-Key Infrastructure (PKI) | A system of policies, roles, certificates, keys, and supporting services used to establish and manage trust in public keys. |
| Root of Trust | A component or mechanism trusted to provide a foundational security function, such as verifying the next stage of software during startup. |
| Secure Boot | A startup mechanism that verifies software against defined trust requirements before allowing it to execute. |
| Secure Provisioning | The controlled process of assigning identities, credentials, keys, and initial security settings to a device. |
| Tamper Resistance | Design features intended to make unauthorized physical access, modification, or interference more difficult. |

| | |
|---|---|
| Trusted Execution Environment (TEE) | An isolated processing environment designed to protect selected code and data from other software running on the same platform. |
| Two-Factor Authentication (2FA) | Authentication requiring two different categories of evidence, such as something known and something possessed. |
| Zeroization | The deliberate removal or destruction of sensitive data, particularly cryptographic secrets, from storage or memory. |

# Drone Security Assessment and Vulnerability Research

| Term | Definition |
| --- | --- |
| Assessment Scope | The systems, interfaces, activities, and boundaries included in an authorized security assessment. |
| Attack Path | A sequence of conditions, weaknesses, and actions through which an unauthorized party could reach a target or achieve an objective. |
| Attack Tree | A structured representation of possible paths an attacker might take to achieve a defined objective. |
| Baseline Testing | Testing performed against a documented reference configuration or expected system behavior. |
| Black-Box Testing | Testing conducted with limited or no access to the target's |

| | |
|---|---|
| | internal design or source code. |
| Code Review | Examination of source code to identify defects, unsafe patterns, and deviations from security requirements. |
| Common Vulnerability Scoring System (CVSS) | A standardized framework for expressing the technical severity of vulnerabilities through defined metrics. It does not replace environment-specific risk analysis. |
| False Negative | A test or detection result that fails to identify a condition that is actually present. |
| False Positive | A result that identifies a potential problem when the reported condition is not actually present as defined. |
| Fuzz Testing | A testing technique that supplies varied, unexpected, or malformed inputs to software to identify crashes, inconsistent behavior, and other defects. |
| Grey-Box Testing | Testing conducted with partial |

| | |
|---|---|
| | knowledge of the target's internal design, source code, or configuration. |
| Penetration Testing | An authorized assessment that attempts to demonstrate whether security weaknesses can be used to compromise defined targets or objectives. |
| Proof of Concept (PoC) | A limited demonstration that a particular technical claim or suspected weakness can be reproduced under specified conditions. |
| Reproducibility | The ability of another qualified person to repeat a test or independently verify its results using documented methods and conditions. |
| Red Team Exercise | An authorized activity designed to evaluate defenses by simulating selected adversary behaviors within defined rules and safety limits. |
| Regression Testing | Testing performed after a change to verify that previously working behavior remains |

| | |
|---|---|
| | correct and known defects have not returned. |
| Rules of Engagement | The agreed conditions, limits, permissions, and procedures governing a security assessment or test. |
| Root Cause | The underlying condition that produced or enabled a defect, incident, or failure. |
| Threat Modeling | A structured process for identifying assets, trust boundaries, potential threats, weaknesses, and appropriate mitigations. |
| Vulnerability Assessment | A process for identifying, evaluating, and prioritizing weaknesses in systems or configurations. |
| White-Box Testing | Testing conducted with substantial knowledge of the target's internal implementation, source code, or design. |

# Monitoring, Incident Response, and Recovery

| Term | Definition |
| --- | --- |
| Alert | A notification that a monitored condition may require attention or investigation. |
| Anomaly Detection | The identification of behavior that differs significantly from an established baseline or expected operating pattern. |
| Audit Log | A record of relevant actions and events used to support accountability, investigation, or compliance review. |
| Chain of Custody | Documentation recording how evidence was collected, handled, transferred, stored, and accessed. |
| Containment | Actions taken to limit the spread, scope, or consequences of a security incident. |

| | |
|---|---|
| Cybersecurity Incident | An event that compromises or threatens the confidentiality, integrity, or availability of systems or information, or violates established security policies. |
| Digital Forensics | The systematic collection, preservation, examination, and interpretation of digital evidence. |
| Incident Response | The coordinated process of preparing for, detecting, analyzing, containing, resolving, and learning from security incidents. |
| Indicators of Compromise (IoCs) | Observable evidence that may indicate a system has been compromised, such as unexpected files, suspicious connections, or unauthorized configuration changes. |
| Mean Time to Detect (MTTD) | A measurement of the average time taken to identify an incident or relevant failure after it begins, under a defined measurement method. |

| | |
|---|---|
| Mean Time to Recover (MTTR) | A measurement of the average time required to restore a system or service after a failure or incident. The exact meaning depends on the organization's definition of recovery. |
| Post-Incident Review | A structured examination of an incident to identify causes, evaluate response effectiveness, and define improvements. |
| Recovery Procedure | A documented process for restoring a system to a known, acceptable, and operationally safe condition. |
| Security Event | An observed occurrence relevant to security, whether or not it ultimately proves to be an incident. |
| Security Information and Event Management (SIEM) | A system that collects, correlates, and analyzes security-related records from multiple sources to support detection and investigation. |
| Threat Intelligence | Information about threats, threat actors, methods, |

indicators, and context used to inform security decisions.

# Risk Management, Governance, and Professional Practice

| Term | Definition |
| --- | --- |
| Acceptance of Risk | A documented decision by an authorized party to retain a known risk rather than eliminate or reduce it immediately. |
| Asset | Anything of value that requires protection, including hardware, software, information, credentials, services, and operational capabilities. |
| Compensating Control | An alternative safeguard used to reduce risk when the preferred control cannot be implemented or is not sufficient. |
| Confidentiality, Integrity, and Availability (CIA) | A foundational security model describing three objectives: preventing unauthorized disclosure, preserving correctness and trustworthiness, and ensuring authorized access when needed. |

| | |
|---|---|
| Continuous Improvement | An ongoing process of evaluating performance, identifying weaknesses, implementing changes, and checking whether those changes improve security. |
| Corrective Action | A measure intended to remove the cause of an identified defect, weakness, or nonconformity. |
| Due Diligence | Reasonable investigation and evaluation performed before making a decision or accepting a responsibility. |
| Governance | The policies, roles, decision-making structures, and oversight mechanisms used to direct and control an organization or system. |
| Impact Analysis | Evaluation of the consequences that could result from a failure, vulnerability, or security incident. |
| Likelihood | An estimate of how probable a defined event or scenario is under specified conditions. |

| | |
|---|---|
| Mitigation | A measure that reduces the likelihood, impact, or scope of a risk. |
| Residual Risk | The risk that remains after safeguards and mitigation measures have been applied. |
| Responsible Disclosure | The coordinated communication of a vulnerability to the relevant party so it can be assessed and addressed while unnecessary harm is minimized. |
| Risk Assessment | The process of identifying risks, analyzing their likelihood and consequences, and determining appropriate responses. |
| Risk Owner | The person or organizational role accountable for evaluating, treating, or formally accepting a particular risk. |
| Security Assurance | Evidence-based confidence that a system's security requirements and controls are appropriately designed and operating as intended within a defined scope. |

| | |
|---|---|
| Security Maintenance | Ongoing work to preserve and improve security through updates, configuration reviews, credential management, monitoring, and verification. |
| Security Policy | A formal statement defining security requirements, responsibilities, and expected behavior. |
| Threat | A potential cause of an unwanted incident that could harm a system, organization, person, or asset. |
| Threat Actor | An individual, group, or entity capable of carrying out actions that may threaten a system. |
| Trust Boundary | A point or region where assumptions about identity, authority, data integrity, or system control change. |
| Vulnerability Disclosure Policy | A documented policy explaining how security researchers and other parties can report suspected vulnerabilities to an organization. |

# Testing, Development, and Software Supply Chain

| Term | Definition |
| --- | --- |
| Build Pipeline | An automated or controlled sequence of steps used to compile, package, test, and prepare software for release. |
| Continuous Integration (CI) | A development practice in which changes are integrated frequently and subjected to automated or repeatable checks. |
| Continuous Delivery (CD) | A software engineering practice that keeps changes in a releasable state through controlled build, testing, and delivery processes. |
| Dependency Management | The process of identifying, selecting, updating, and controlling software components required by a project. |
| Firmware Provenance | Documented information about where firmware originated, how |

| | |
|---|---|
| | it was built, and how its identity and integrity can be verified. |
| Patch | A software change intended to correct a defect, address a vulnerability, or modify existing behavior. |
| Reproducible Build | A build process designed so that the same source and controlled inputs produce the same output, or an output that can be independently verified against the expected result. |
| Release Management | The controlled process of approving, distributing, documenting, and maintaining software or firmware releases. |
| Secure Development Lifecycle | A development approach that incorporates security requirements, design review, testing, release controls, and maintenance throughout a product's life. |
| Signed Update | A software or firmware package accompanied by a digital signature that can be checked against a trusted verification key. |

| | |
|---|---|
| Software Supply Chain | The people, processes, tools, code, libraries, build systems, and distribution mechanisms involved in producing and delivering software. |
| Test Environment | A controlled arrangement of hardware, software, configuration, and supporting services used to evaluate system behavior. |
| Version Control | A system for recording and managing changes to source code, configuration, documents, or other project artifacts. |

# Operational and Regulatory Terms

| Term | Definition |
| --- | --- |
| Beyond Visual Line of Sight (BVLOS) | An operation in which the remote pilot cannot continuously maintain unaided visual line of sight with the aircraft, subject to applicable definitions and operating rules. |
| Command and Control (C2) | The communication and control functions used to transmit instructions to an aircraft and receive information needed to manage its operation. |
| Detect and Avoid (DAA) | Technologies and procedures intended to detect relevant hazards, such as other aircraft, and support actions to maintain safe separation. |
| Flight Termination | A defined action or mechanism intended to end a flight under specified circumstances. Its implementation must be evaluated carefully because termination can itself create hazards. |

| | |
|---|---|
| Operational Risk Assessment | Evaluation of risks arising from the planned operation, including the aircraft, environment, people, procedures, and supporting systems. |
| Remote Pilot | The person responsible for managing or conducting an unmanned aircraft flight in accordance with the applicable operating framework. |
| Unmanned Aircraft (UA) | An aircraft designed to operate without a person onboard. |
| Unmanned Aircraft System (UAS) | The aircraft and the associated equipment, control stations, communication links, and other components needed to conduct its operation. |
| Unmanned Aerial Vehicle (UAV) | A commonly used term for an aircraft that operates without a pilot onboard. Usage varies, and it may refer specifically to the aircraft rather than the complete system. |
| Visual Line of Sight (VLOS) | An operating condition in which |

the remote pilot or designated observer maintains the required unaided visual contact with the aircraft, as defined by applicable rules.