OWASP API Security Top 10 2023

WWW.DEVSECOPSGUIDES.COM

DevSecOpsGuides.com

OWASP API Security Top 10 2023

API1:2023 - Broken Object Level Authorization

This vulnerability arises due to the lack of permission checks on resources and objects. An attacker can exploit this to gain access to resources and data of user groups and the system. Permission checks at the object level should be considered in any function that accesses a data resource using an identifier from the user.

Example:

GET request to retrieve the details of a product with the product ID:

```
GET /api/products/{product_id}
```

Noncompliant Code(.NET):

```
// Non-compliant code
public class UserController : ApiController
{
    [HttpGet]
    public User GetUser(int userId)
    {
        User user = UserRepository.GetUserById(userId);
        return user;
    }

    [HttpPut]
    public IHttpActionResult UpdateUser(User user)
    {
        UserRepository.UpdateUser(user);
        return Ok();
    }
}
```

Compliant Code(.NET):

```
// Compliant code
public class UserController : ApiController
{
    [HttpGet]
    [Authorize(Roles = "Admin")]
    public User GetUser(int userId)
    {
        User user = UserRepository.GetUserById(userId);
        return user;
    }
    [HttpPut]
    [Authorize(Roles = "Admin")]
    public IHttpActionResult UpdateUser(User user)
    {
        UserRepository.UpdateUser(user);
        return Ok();
    }
}
```

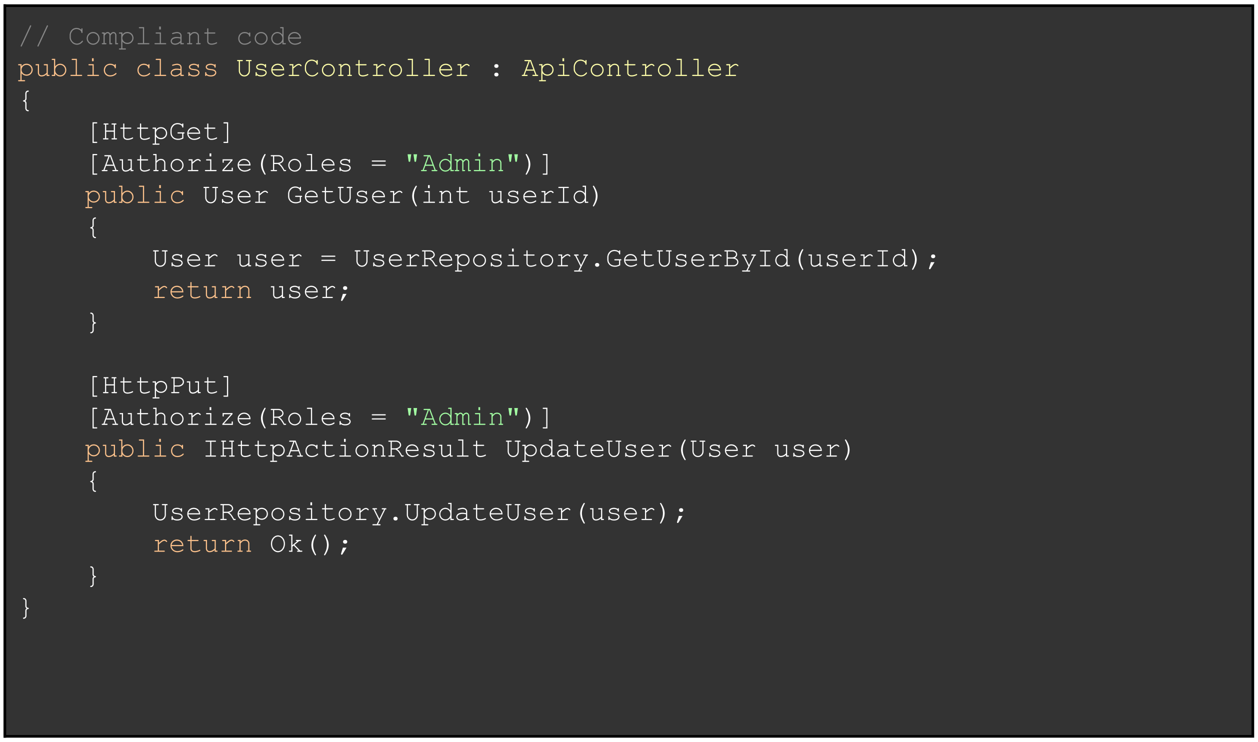
Noncompliant Code(Java):

```
// Non-compliant code
@RestController
public class UserController {

    @GetMapping("/users/{userId}")
    public User getUser(@PathVariable int userId) {
        User user = UserRepository.getUserById(userId);
        return user;
    }

    @PutMapping("/users/{userId}")
    public ResponseEntity<?> updateUser(@PathVariable int userId,
@RequestBody User user) {
        UserRepository.updateUser(user);
        return ResponseEntity.ok().build();
    }
}
```

Compliant Code(Java):



```
// Compliant code
public class UserController : ApiController
{
    [HttpGet]
    [Authorize(Roles = "Admin")]
    public User GetUser(int userId)
    {
        User user = UserRepository.GetUserById(userId);
        return user;
    }

    [HttpPut]
    [Authorize(Roles = "Admin")]
    public IHttpActionResult UpdateUser(User user)
    {
        UserRepository.UpdateUser(user);
        return Ok();
    }
}
```

General Prevention Suggestions:

In any function that accesses a data resource using a user's identifier, consider the necessary checks for object-level permission. Ensure that the user is authorized to access this resource.

Validate identifiers and permissions in each request. Make sure that a user attempting to access a specific object is authorized to do so.

Protect the way the user's identifier is sent in requests. Use secure methods for transmitting and storing the identifier, such as using authentication tokens.

API2:2023 - Broken Authentication

In this vulnerability, due to insufficient security mechanisms for user authentication to access resources, there's the possibility of disruption and unauthorized access to protected information by an attacker.

Example:

POST request for user login using login details:

```
POST /api/login

Body:

{

  "username": "exampleuser",

  "password": "secretpassword"

}
```

Noncompliant Code(.NET):

```
// Non-compliant code
[ApiController]
[Route("api/[controller]")]
```

```
public class UserController : ControllerBase
{
    [HttpPost]
    public IActionResult Login(string username, string password)
    {
        if (AuthenticateUser(username, password))
        {
            // Generate and return authentication token
            var token = GenerateAuthToken(username);
            return Ok(token);
        }
        else
        {
            return Unauthorized();
        }
    }

    [HttpGet]
    public IActionResult GetUserData(int userId)
    {
        // Retrieve user data from the database
        var userData = Database.GetUserById(userId);

        // Return user data
        return Ok(userData);
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
// Compliant code
[ApiController]
```

```
[Route("api/[controller]")]
public class UserController : ControllerBase
{
    private readonly IUserService _userService;
    private readonly IAuthenticationService _authenticationService;

    public UserController(IUserService userService,
IAuthenticationService authenticationService)
    {
        _userService = userService;
        _authenticationService = authenticationService;
    }

    [HttpPost]
    public IActionResult Login(LoginModel loginModel)
    {
        if (_authenticationService.AuthenticateUser(loginModel.Username,
loginModel.Password))
        {
            // Generate and return authentication token
            var token =
_authenticationService.GenerateAuthToken(loginModel.Username);
            return Ok(token);
        }
        else
        {
            return Unauthorized();
        }
    }

    [HttpGet]
    [Authorize]
    public IActionResult GetUserData(int userId)
    {
        // Retrieve the authenticated user's identity
        var identity = HttpContext.User.Identity as ClaimsIdentity;
        if (identity != null)
        {
            // Get the user ID from the authentication token
            var userIdFromToken = identity.FindFirst("UserId")?.Value;

            if (!string.IsNullOrEmpty(userIdFromToken) &&
userIdFromToken == userId.ToString())
            {
                // Retrieve user data from the database
                var userData = _userService.GetUserData(userId);
                return Ok(userData);
            }
        }

        return Unauthorized();
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username,
@RequestParam String password) {
        if (userService.authenticateUser(username, password)) {
            // Generate and return authentication token
            String token = generateAuthToken(username);
            return ResponseEntity.ok(token);
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }

    @GetMapping("/{userId}")
    public ResponseEntity<User> getUserData(@PathVariable int userId) {
        // Retrieve user data from the database
        User user = userService.getUserById(userId);

        // Return user data
        return ResponseEntity.ok(user);
    }

    // Other methods...
}
```

Compliant Code(Java):

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @Autowired
    private AuthenticationService authenticationService;

    @PostMapping("/login")
    public ResponseEntity<String> login(@RequestParam String username, 
@RequestParam String password) {
        if (authenticationService.authenticateUser(username, password)) 
{
            // Generate and return authentication token
            String token = 
authenticationService.generateAuthToken(username);
            return ResponseEntity.ok(token);
        } else {
            return 
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }

    @GetMapping("/{userId}")
    public ResponseEntity<User> getUserData(@PathVariable int userId,
@RequestHeader("Authorization") String authToken) {
        if (authenticationService.validateAuthToken(authToken)) {
            // Retrieve user data from the database
            User user = userService.getUserById(userId);

            // Return user data
            return ResponseEntity.ok(user);
        } else {
            return
ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();
        }
    }

    // Other methods...
}
```

General Prevention Suggestions:

Use strong and standard authentication mechanisms, such as JWT (JSON Web Tokens) or OAuth.

Use robust encryption methods for storing and transmitting sensitive information, like connection encryption (TLS/SSL).

Properly verify authentication information and ensure that every authenticated request comes from a legitimate user.

Carefully ensure that authentication details (like passwords) are securely encrypted during transmission or storage on the server.

Implement a limit on the number of unsuccessful login attempts and temporarily lock user accounts after a certain number of failed tries.

API3:2023 - Broken Object Property Level Authorization

In this vulnerability, due to the lack of data model verification in the request and response, an attacker has the ability to extract or perform CRUD operations on the related methods. This issue arises due to the lack of proper validation of access permissions to object attributes, leading to information disclosure or disruption in requests.

Example:

PUT request to update a feature of an item:

PUT /api/items/{item_id}

Body:

{

  "name": "Updated Item Name",

  "price": 10.99,

  "is_available": true

}

Noncompliant Code(.NET):

```
[Route("api/items")]
public class ItemController : ControllerBase
{
    private readonly IItemService _itemService;

    public ItemController(IItemService itemService)
    {
        _itemService = itemService;
    }

    [HttpGet("{itemId}")]
    public IActionResult GetItem(int itemId)
    {
        // Retrieve the item from the database
        Item item = _itemService.GetItem(itemId);

        // Return the item without any authorization check
        return Ok(item);
    }

    [HttpPut("{itemId}")]
    public IActionResult UpdateItem(int itemId, [FromBody] Item 
updatedItem)
    {
        // Retrieve the existing item from the database
        Item existingItem = _itemService.GetItem(itemId);

        // Update only the allowed properties
        existingItem.Name = updatedItem.Name;
        existingItem.Price = updatedItem.Price;
        existingItem.IsAvailable = updatedItem.IsAvailable;

        // Save the changes to the database
        _itemService.UpdateItem(existingItem);

        // Return a success response
        return Ok();
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[Route("api/items")]
public class ItemController : ControllerBase
{
    private readonly IItemService _itemService;

    public ItemController(IItemService itemService)
    {
        _itemService = itemService;
    }

    [HttpGet("{itemId}")]
    public IActionResult GetItem(int itemId)
    {
        // Retrieve the item from the database
        Item item = _itemService.GetItem(itemId);

        // Check if the user is authorized to access the item
        if (!IsUserAuthorized(item))
        {
            return Forbid();
        }

        // Return the item
        return Ok(item);
    }

    [HttpPut("{itemId}")]
    public IActionResult UpdateItem(int itemId, [FromBody] Item
updatedItem)
    {
        // Retrieve the existing item from the database
        Item existingItem = _itemService.GetItem(itemId);

        // Check if the user is authorized to update the item properties
        if (!IsUserAuthorized
```

Noncompliant Code(Java):

```java
@RestController
@RequestMapping("/api/items")
public class ItemController {

    private final ItemService itemService;

    public ItemController(ItemService itemService) {
        this.itemService = itemService;
    }

    @GetMapping("/{itemId}")
    public Item getItem(@PathVariable int itemId) {
        // Retrieve the item from the database
        Item item = itemService.getItem(itemId);

        // Return the item without any authorization check
        return item;
    }

    @PutMapping("/{itemId}")
    public void updateItem(@PathVariable int itemId, @RequestBody Item
updatedItem) {
        // Retrieve the existing item from the database
        Item existingItem = itemService.getItem(itemId);

        // Update only the allowed properties
        existingItem.setName(updatedItem.getName());
        existingItem.setPrice(updatedItem.getPrice());
        existingItem.setAvailable(updatedItem.isAvailable());
        // Save the changes to the database
        itemService.updateItem(existingItem);
    }

    // Other methods...
}
```

Compliant Code(Java):

```
@RestController
@RequestMapping("/api/items")
public class ItemController {

    private final ItemService itemService;

    public ItemController(ItemService itemService) {
        this.itemService = itemService;
    }

    @GetMapping("/{itemId}")
    public ResponseEntity<Item> getItem(@PathVariable int itemId) {
        // Retrieve the item from the database
        Item item = itemService.getItem(itemId);

        // Check if the user is authorized to access the item
        if (!isUserAuthorized(item)) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }
        // Return the item
        return ResponseEntity.ok(item);
    }

    @PutMapping("/{itemId}")
    public ResponseEntity<Void> updateItem(@PathVariable int itemId,
@RequestBody Item updatedItem) {
        // Retrieve the existing item from the database
        Item existingItem = itemService.getItem(itemId);

        // Check if the user is authorized to update the item properties
        if (!isUserAuthorized(existingItem)) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
        }

        // Only update the allowed properties
        existingItem.setName(updatedItem.getName());
        existingItem.setPrice(updatedItem.getPrice());
        existingItem.setAvailable(updatedItem.isAvailable());

        // Save the changes to the database
        itemService.updateItem(existingItem);

        // Return a success response
        return ResponseEntity.ok().build();
    }

    private boolean isUserAuthorized(Item item) {
        // Implement your authorization logic here
        // Check if the user has the necessary permissions to access the
item
        // Return true if authorized, false otherwise
    }

    // Other methods...
}
```

General Prevention Suggestions:

When creating or updating objects, ensure that access permissions for features are set at the correct level.

Validate user input data and only accept them if they have authorized access to the relevant features.

Use robust and secure mechanisms for defining and managing permissions and roles in the system, like RBAC (Role-Based Access Control).

Limit user access to object features based on business needs and the Principle of Least Privilege.

Conduct regular security tests on APIs and the system to ensure that all required permissions and validations have been correctly implemented.

API4:2023 - Unrestricted Resource Consumption

Due to this vulnerability, the attacker can disrupt the API service delivery status due to the lack of limitations on requests, leading to errors resulting from insufficient resources for processing.

Example:

POST request to send an SMS to a specific mobile number:

```
POST /api/sms/send

Body:

{

  "phone_number": "1234567890",

  "message": "Hello, this is a test message."

}
```

Noncompliant Code(.NET):

```
[ApiController]
[Route("api/resource")]
public class ResourceController : ControllerBase
```

```
{
    private readonly ResourceService _resourceService;

    public ResourceController(ResourceService resourceService)
    {
        _resourceService = resourceService;
    }

    [HttpPost]
    public IActionResult ProcessResource(ResourceRequest request)
    {
        // Process the resource request
        string result = _resourceService.Process(request);

        // Return the result
        return Ok(result);
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[ApiController]
[Route("api/resource")]
```

```
public class ResourceController : ControllerBase
{
    private readonly ResourceService _resourceService;

    public ResourceController(ResourceService resourceService)
    {
        _resourceService = resourceService;
    }

    [HttpPost]
    public IActionResult ProcessResource(ResourceRequest request)
    {
        // Validate the resource request
        if (!IsValidRequest(request))
        {
            return BadRequest();
        }

        // Process the resource request with resource consumption limits
        bool success = _resourceService.ProcessWithLimits(request);

        // Check if the resource consumption was successful
        if (!success)
        {
            return StatusCode((int)HttpStatusCode.TooManyRequests);
        }

        // Return the result
        return Ok("Resource processed successfully");
    }

    private bool IsValidRequest(ResourceRequest request)
    {
        // Implement your validation logic here
        // Check if the request is valid
        // Return true if valid, false otherwise
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```
@RestController
@RequestMapping("/api")
public class ResourceController {

    private final ResourceService resourceService;

    public ResourceController(ResourceService resourceService) {
        this.resourceService = resourceService;
    }

    @PostMapping("/resource")
    public ResponseEntity<String> processResource(@RequestBody
ResourceRequest request) {
        // Process the resource request
        String result = resourceService.process(request);

        // Return the result
        return ResponseEntity.ok(result);
    }

    // Other methods...
}
```

Compliant Code(Java):

```java
@RestController
@RequestMapping("/api")
public class ResourceController {

    private final ResourceService resourceService;

    public ResourceController(ResourceService resourceService) {
        this.resourceService = resourceService;
    }

    @PostMapping("/resource")
    public ResponseEntity<String> processResource(@RequestBody
ResourceRequest request) {
        // Validate the resource request
        if (!isValidRequest(request)) {
            return ResponseEntity.badRequest().build();
        }

        // Process the resource request with resource consumption limits
        boolean success = resourceService.processWithLimits(request);

        // Check if the resource consumption was successful
        if (!success) {
            return
ResponseEntity.status(HttpStatus.TOO_MANY_REQUESTS).build();
        }

        // Return the result
        return ResponseEntity.ok("Resource processed successfully");
    }

    private boolean isValidRequest(ResourceRequest request) {
        // Implement your validation logic here
        // Check if the request is valid
        // Return true if valid, false otherwise
    }

    // Other methods...
}
```

General Prevention Suggestions:

Limit the resources consumed by each API request, such as bandwidth limitation, the number of requests in a specific time interval, and the maximum number of SMS messages or phone calls.

Review and validate API requests based on the allowable ceiling for resource consumption and apply necessary limitations.

Use traffic limiting and bandwidth control mechanisms like Advanced Network Limiting to manage the resources consumed by each user or service.

Monitor and log resource consumption to detect suspicious patterns and take more detailed validations when necessary.

Conduct Load Testing and evaluate the system's resource performance to detect and prevent issues from improper resource consumption.

API5:2023 - Broken Function Level Authorization

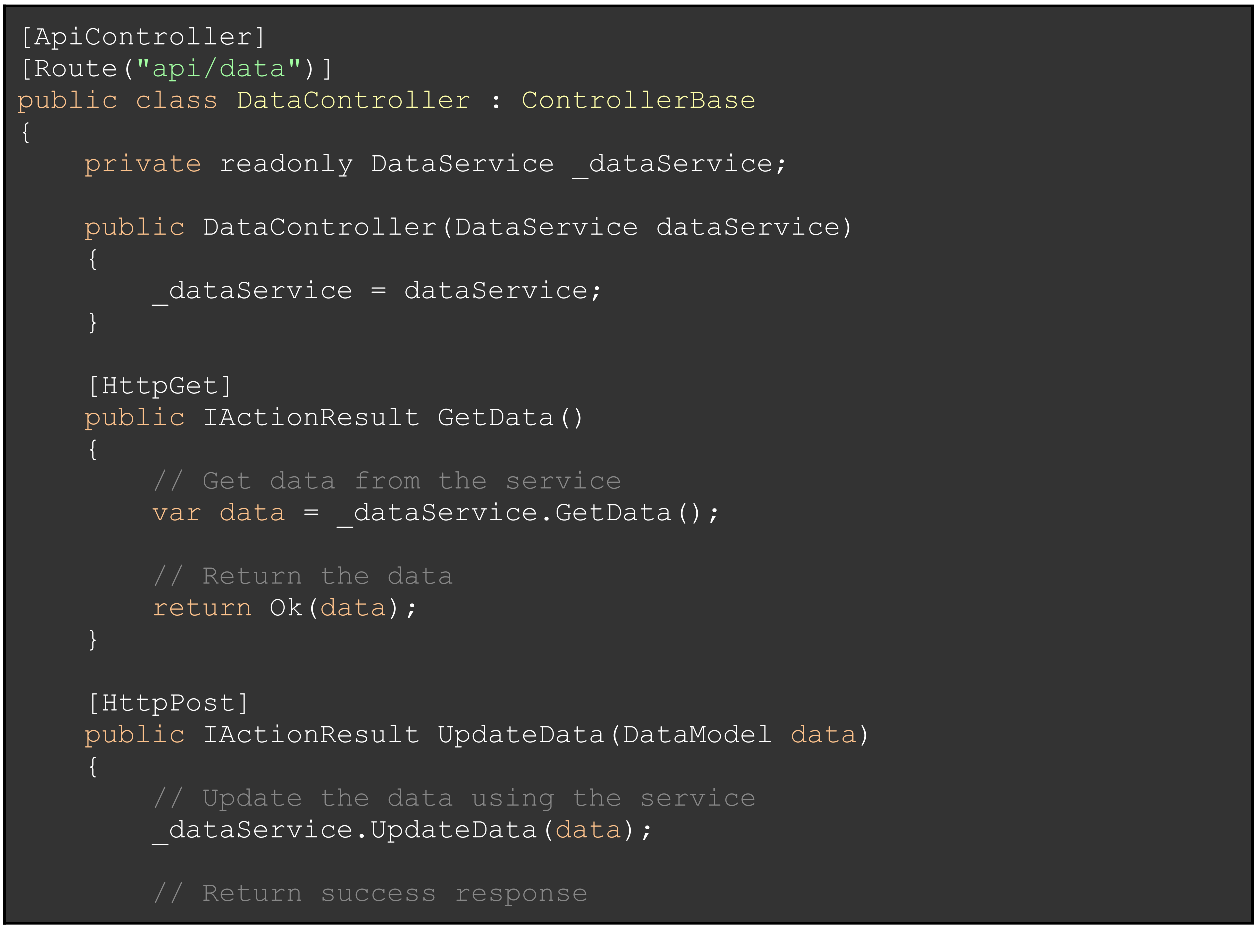
Due to the lack of enforcing access control policies with hierarchical permissions, the attacker can invoke and execute unauthorized requests from the allowed Endpoint to access other users' resources and/or managerial functionalities.

Example:

DELETE request to delete a comment with the comment ID:

DELETE /api/comments/{comment_id}

Noncompliant Code(.NET):



```
[ApiController]
[Route("api/data")]
public class DataController : ControllerBase
{
    private readonly DataService _dataService;

    public DataController(DataService dataService)
    {
        _dataService = dataService;
    }

    [HttpGet]
    public IActionResult GetData()
    {
        // Get data from the service
        var data = _dataService.GetData();

        // Return the data
        return Ok(data);
    }

    [HttpPost]
    public IActionResult UpdateData(DataModel data)
    {
        // Update the data using the service
        _dataService.UpdateData(data);

        // Return success response
        return Ok("Data updated successfully");
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[ApiController]
[Route("api/data")]
[Authorize]
public class DataController : ControllerBase
{
    private readonly DataService _dataService;

    public DataController(DataService dataService)
    {
        _dataService = dataService;
    }

    [HttpGet]
    [Authorize(Roles = "ReadAccess")]
    public IActionResult GetData()
    {
        // Get the user's identity
        var identity = HttpContext.User.Identity as ClaimsIdentity;

        // Get the user's role
        var role = identity.FindFirst(ClaimTypes.Role)?.Value;
        // Check if the user has the required role for reading data
        if (role != "ReadAccess")
        {
            return Forbid(); // Return 403 Forbidden if the user is not
authorized
        }

        // Get data from the service
        var data = _dataService.GetData();

        // Return the data
        return Ok(data);
    }

    [HttpPost]
    [Authorize(Roles = "WriteAccess")]
    public IActionResult UpdateData(DataModel data)
    {
        // Get the user's identity
        var identity = HttpContext.User.Identity as ClaimsIdentity;

        // Get the user's role
        var role = identity.FindFirst(ClaimTypes.Role)?.Value;

        // Check if the user has the required role for updating data
        if (role != "WriteAccess")
        {
            return Forbid(); // Return 403 Forbidden if the user is not
authorized
        }

        // Update the data using the service
        _dataService.UpdateData(data);

        // Return success response
        return Ok("Data updated successfully");
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```java
@RestController
@RequestMapping("/api/data")
public class DataController {
    private final DataService dataService;

    public DataController(DataService dataService) {
        this.dataService = dataService;
    }

    @GetMapping
    public ResponseEntity<List<Data>> getData() {
        // Get data from the service
        List<Data> data = dataService.getData();

        // Return the data
        return ResponseEntity.ok(data);
    }

    @PostMapping
    public ResponseEntity<String> updateData(@RequestBody Data data) {
        // Update the data using the service
        dataService.updateData(data);

        // Return success response
        return ResponseEntity.ok("Data updated successfully");
    }

    // Other methods...
}
```

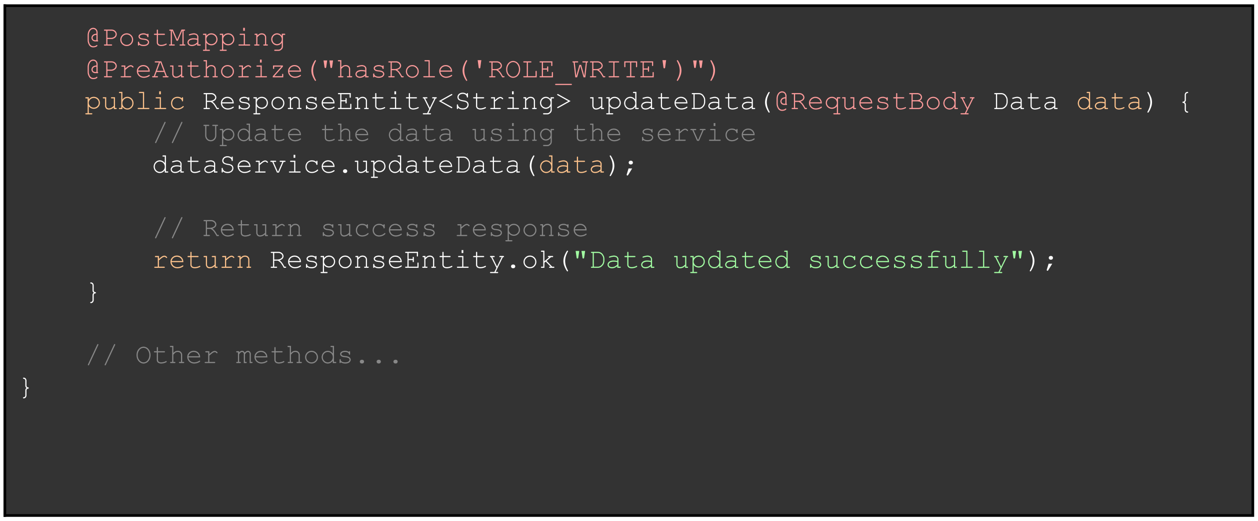
Compliant Code(Java):

```
@RestController
@RequestMapping("/api/data")
public class DataController {
    private final DataService dataService;

    public DataController(DataService dataService) {
        this.dataService = dataService;
    }

    @GetMapping
    @PreAuthorize("hasRole('ROLE_READ')")
    public ResponseEntity<List<Data>> getData() {
        // Get data from the service
        List<Data> data = dataService.getData();

        // Return the data
        return ResponseEntity.ok(data);
    }
```



```
    @PostMapping
    @PreAuthorize("hasRole('ROLE_WRITE')")
    public ResponseEntity<String> updateData(@RequestBody Data data) {
        // Update the data using the service
        dataService.updateData(data);

        // Return success response
        return ResponseEntity.ok("Data updated successfully");
    }

    // Other methods...
}
```

General prevention suggestions:

Complete validation in every API function based on access levels and user roles.

Use multi-level access permission systems and apply access levels to various resources.

Properly segregate between managerial and regular functionalities and enforce appropriate access policies for each.

Examine permissions in every function and validate user access at runtime.

Utilize frameworks and libraries for managing user access and implement more complex access policies like RBAC (Role-Based Access Control) or ABAC (Attribute-Based Access Control).

API6:2023 - Unrestricted Access to Sensitive Business Flows

Due to this vulnerability, an attacker has the ability to exploit the legitimate functions of the application for illicit purposes because of the app's capabilities.

Example:

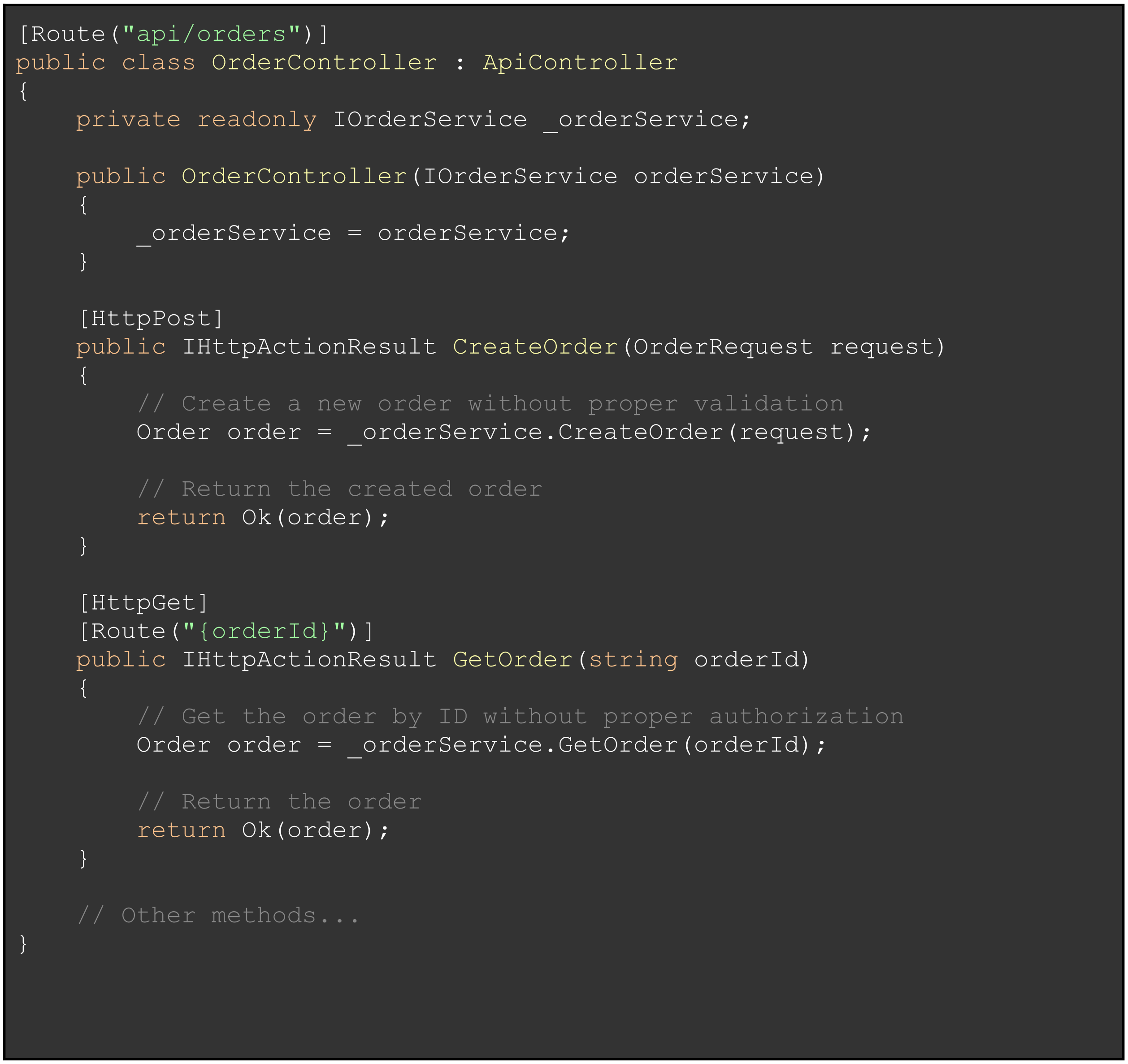
POST request to purchase an airplane ticket by providing passenger details.

```
POST /api/tickets/buy

Body:

{
  "passenger_name": "John Doe",
  "flight_number": "AB123",
  "departure_date": "2023-07-01"
}
```

Noncompliant Code(.NET):



```
[Route("api/orders")]
public class OrderController : ApiController
{
    private readonly IOrderService _orderService;

    public OrderController(IOrderService orderService)
    {
        _orderService = orderService;
    }

    [HttpPost]
    public IHttpActionResult CreateOrder(OrderRequest request)
    {
        // Create a new order without proper validation
        Order order = _orderService.CreateOrder(request);

        // Return the created order
        return Ok(order);
    }

    [HttpGet]
    [Route("{orderId}")]
    public IHttpActionResult GetOrder(string orderId)
    {
        // Get the order by ID without proper authorization
        Order order = _orderService.GetOrder(orderId);

        // Return the order
        return Ok(order);
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[Route("api/orders")]
public class OrderController : ApiController
{
    private readonly IOrderService _orderService;

    public OrderController(IOrderService orderService)
    {
        _orderService = orderService;
    }

    [HttpPost]
    [Authorize(Roles = "Admin")]
    public IHttpActionResult CreateOrder(OrderRequest request)
    {
        // Validate the request and create a new order with proper
authorization
        Order order = _orderService.CreateOrder(request);

        // Return the created order
        return Ok(order);
    }

    [HttpGet]
    [Route("{orderId}")]
    [Authorize(Roles = "User")]
    public IHttpActionResult GetOrder(string orderId)
    {
        // Authorize the user's access to the order
        // Only users with the "User" role can access the order
        Order order = _orderService.GetOrder(orderId);

        // Return the order
        return Ok(order);
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```
@RestController
@RequestMapping("/api/orders")
public class OrderController {
    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
    }

    @PostMapping
    public ResponseEntity<Order> createOrder(@RequestBody OrderRequest
request) {
        // Create a new order without proper validation
        Order order = orderService.createOrder(request);

        // Return the created order
        return ResponseEntity.ok(order);
    }

    @GetMapping("/{orderId}")
    public ResponseEntity<Order> getOrder(@PathVariable String orderId)
{
        // Get the order by ID without proper authorization
        Order order = orderService.getOrder(orderId);

        // Return the order
        return ResponseEntity.ok(order);
    }

    // Other methods...
}
```

Compliant Code(Java):

```
@RestController
@RequestMapping("/api/orders")
public class OrderController {
    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
    }

    @PostMapping
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public ResponseEntity<Order> createOrder(@RequestBody OrderRequest
request) {
        // Validate the request and create a new order with proper
authorization
        Order order = orderService.createOrder(request);
        // Return the created order
        return ResponseEntity.ok(order);
    }

    @GetMapping("/{orderId}")
    @PreAuthorize("hasRole('ROLE_USER') or hasPermission(#orderId,
'READ')")
    public ResponseEntity<Order> getOrder(@PathVariable String orderId)
{
        // Authorize the user's access to the order
        // Only users with ROLE_USER or permission to read the order can
access it
        Order order = orderService.getOrder(orderId);

        // Return the order
        return ResponseEntity.ok(order);
    }

    // Other methods...
}
```

General prevention recommendations:

Implement authentication and user validation mechanisms before accessing sensitive business flows.

Carefully inspect and validate user data and inputs, including the validity of dates and input formats.

Apply limitations and logical rules for accessing sensitive business flows.

Use logging and monitoring systems to detect and track suspicious or inappropriate activities in business flows.

Provide and utilize interfaces (API Gateways) that facilitate control and management of access to business flows.

API7:2023 - Server Side Request Forgery

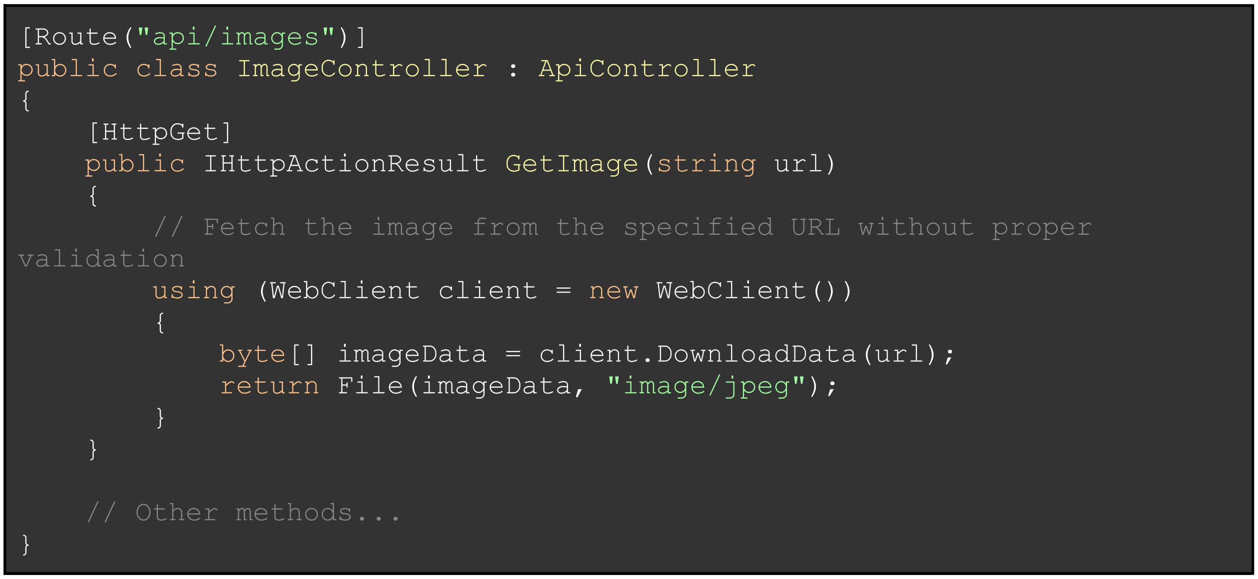
Through this vulnerability, the attacker has the ability to forge requests on the server side and send fraudulent requests to an authorized destination.

Example:

A GET request to retrieve an image from a specific URL:

GET /api/image?url=http://malicious-website.com/malware.jpg

Noncompliant Code(.NET):



```
[Route("api/images")]
public class ImageController : ApiController
{
    [HttpGet]
    public IHttpActionResult GetImage(string url)
    {
        // Fetch the image from the specified URL without proper
validation
        using (WebClient client = new WebClient())
        {
            byte[] imageData = client.DownloadData(url);
            return File(imageData, "image/jpeg");
        }
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[Route("api/images")]
public class ImageController : ApiController
{
    [HttpGet]
    public IHttpActionResult GetImage(string url)
    {
        // Validate and sanitize the URL before fetching the image
        if (!IsValidUrl(url))
        {
            return BadRequest("Invalid URL");
        }

        using (WebClient client = new WebClient())
        {
            byte[] imageData = client.DownloadData(url);
            return File(imageData, "image/jpeg");
        }
    }

    private bool IsValidUrl(string url)
    {
        // Implement URL validation logic here (e.g., whitelist trusted
domains)
        // Return true if the URL is valid, otherwise false
        // Example validation logic:
        return url.StartsWith("http://trusted-domain.com");
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```java
@RestController
@RequestMapping("/api/images")
public class ImageController {

    @GetMapping
    public ResponseEntity<byte[]> getImage(@RequestParam("url") String url) throws IOException {
        // Fetch the image from the specified URL without proper validation
        URL imageUrl = new URL(url);
        byte[] imageData = IOUtils.toByteArray(imageUrl);
        return ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(imageData);
    }

    // Other methods...
}
```

Compliant Code(Java):

```
@RestController
@RequestMapping("/api/images")
public class ImageController {

    @GetMapping
    public ResponseEntity<byte[]> getImage(@RequestParam("url") String
url) throws IOException {
        // Validate and sanitize the URL before fetching the image
        if (!isValidUrl(url)) {
            return ResponseEntity.badRequest().build();
        }

        URL imageUrl = new URL(url);
        byte[] imageData = IOUtils.toByteArray(imageUrl);
        return
ResponseEntity.ok().contentType(MediaType.IMAGE_JPEG).body(imageData);
    }

    private boolean isValidUrl(String url) {
        // Implement URL validation logic here (e.g., whitelist trusted
domains)
        // Return true if the URL is valid, otherwise false
        // Example validation logic:
        return url.startsWith("http://trusted-domain.com");
    }

    // Other methods...
}
```

General prevention suggestions:

Before sending a request to a specific URL, thoroughly check and validate the URI and the target source.

Limit the capability to retrieve information from external sources and restrict access to remote URLs.

Use a Whitelist to only allow valid addresses and access to them.

Validate and filter user inputs and parameters related to the utilized URL before using them in the request.

Employ network restrictions, such as firewalls, to limit access to external resources.

Train the development team to properly evaluate and validate URIs before using them in requests.

API8:2023 - Security Misconfiguration

Due to incorrect configurations or improper management of related settings, an attacker can exploit default or incorrect configurations.

Example:

GET request to retrieve system settings:

```
GET /api/configurations
```

Noncompliant Code(.NET):

```
using System.Web.Http;

namespace MyAPI.Controllers
{
    public class UserController : ApiController
    {
        // GET api/user/{id}
        public IHttpActionResult GetUser(int id)
        {
            // Fetch user data from the database without proper access
control
            var user = Database.GetUser(id);
            return Ok(user);
        }

        // Other methods...
    }
}
```

Compliant Code(.NET):

```
using System.Web.Http;
using Microsoft.AspNetCore.Authorization;
```

```
namespace MyAPI.Controllers
{
    [Authorize] // Apply authorization to the controller
    public class UserController : ApiController
    {
        // GET api/user/{id}
        [Authorize(Roles = "Admin")] // Restrict access to authorized users with the "Admin" role
        public IHttpActionResult GetUser(int id)
        {
            // Fetch user data from the database only if the user has the "Admin" role
            var user = Database.GetUser(id);
            return Ok(user);
        }

        // Other methods...
    }
}
```

Noncompliant Code(Java):

```
@RestController
public class UserController {

    @Autowired
    private UserRepository userRepository;

    // GET /user/{id}
```

```java
    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public User getUser(@PathVariable int id) {
        // Fetch user data from the database without proper access control
        User user = userRepository.findById(id);
        return user;
    }

    // Other methods...
}
```

Compliant Code(Java):

```java
@RestController
public class UserController {

    @Autowired
    private UserRepository userRepository;

    // GET /user/{id}
    @PreAuthorize("hasRole('ADMIN')") // Restrict access to users with the "ADMIN" role
    @RequestMapping(value = "/user/{id}", method = RequestMethod.GET)
    public User getUser(@PathVariable int id) {
        // Fetch user data from the database only if the user has the "ADMIN" role
        User user = userRepository.findById(id);
        return user;
    }
    // Other methods...
}
```

General Prevention Recommendations:

Before sending a request to a specific URL, thoroughly check and validate the URI and the destination source.

Limit the capability to retrieve information from external sources and restrict the list of allowed access to remote URLs.

Use a Whitelist to only allow access to valid addresses.

Validate and filter user inputs and parameters related to the URL in use before incorporating them into requests.

Use network restrictions, like firewalls, to limit access to external resources.

Train the development team to evaluate and validate the correct URI before using it in requests.

API9:2023 - Improper Inventory Management

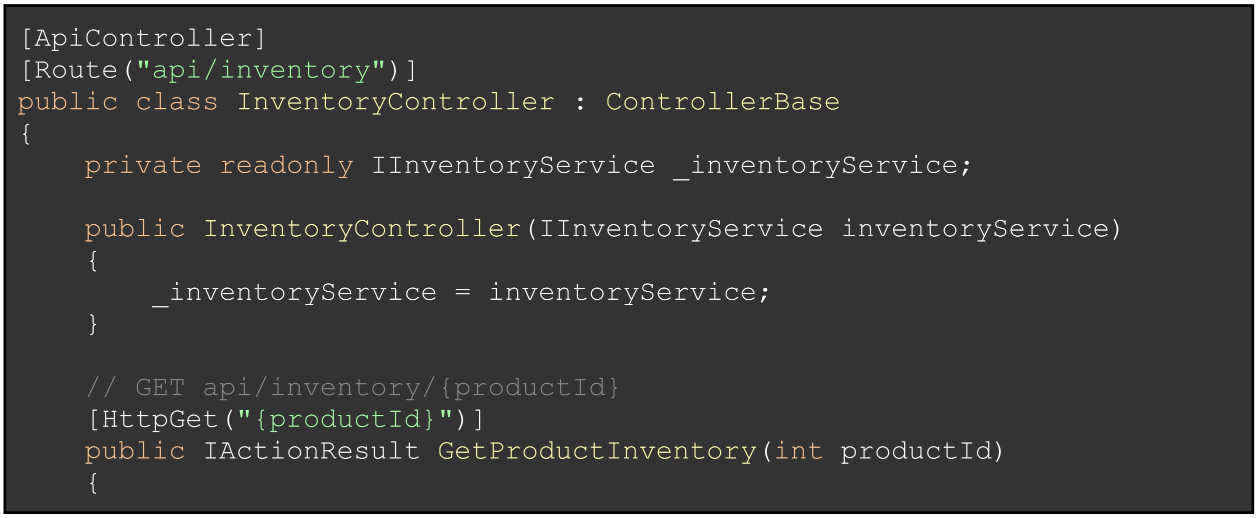
Due to the lack of management of API versions and also the list of capabilities and functionalities for specific use-case scenarios across all functions, an attacker has the possibility to exploit different functionalities in various versions of the application.

Example:

A GET request to retrieve the list of available API versions:

GET /api/versions

Noncompliant Code(.NET):



```
[ApiController]
[Route("api/inventory")]
public class InventoryController : ControllerBase
{
    private readonly IInventoryService _inventoryService;

    public InventoryController(IInventoryService inventoryService)
    {
        _inventoryService = inventoryService;
    }

    // GET api/inventory/{productId}
    [HttpGet("{productId}")]
    public IActionResult GetProductInventory(int productId)
    {
        // Fetch inventory data directly from the database
        var inventory =
_inventoryService.GetInventoryByProductId(productId);
        return Ok(inventory);
    }

    // POST api/inventory
    [HttpPost]
    public IActionResult UpdateProductInventory(InventoryModel
inventory)
    {
        // Update inventory directly in the database
        _inventoryService.UpdateInventory(inventory);
        return Ok();
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[ApiController]
[Route("api/inventory")]
public class InventoryController : ControllerBase
{
    private readonly IInventoryService _inventoryService;

    public InventoryController(IInventoryService inventoryService)
    {
        _inventoryService = inventoryService;
    }

    // GET api/inventory/{productId}
    [HttpGet("{productId}")]
    public IActionResult GetProductInventory(int productId)
    {
        // Fetch inventory data through the inventory service
        var inventory =
_inventoryService.GetProductInventory(productId);

        if (inventory == null)
            return NotFound();

        return Ok(inventory);
    }

    // POST api/inventory
    [HttpPost]
    [Authorize(Roles = "Admin")] // Restrict access to authorized users
with the "Admin" role
    public IActionResult UpdateProductInventory(InventoryModel
inventory)
    {
        // Update inventory through the inventory service
        _inventoryService.UpdateProductInventory(inventory);
        return Ok();
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```
@RestController
@RequestMapping("/api/inventory")
public class InventoryController {

    private final InventoryService inventoryService;

    public InventoryController(InventoryService inventoryService) {
        this.inventoryService = inventoryService;
    }
    // GET /api/inventory/{productId}
    @GetMapping("/{productId}")
    public ResponseEntity<Inventory> getProductInventory(@PathVariable
int productId) {
        // Fetch inventory data directly from the database
        Inventory inventory =
inventoryService.getInventoryByProductId(productId);
        return ResponseEntity.ok(inventory);
    }

    // POST /api/inventory
    @PostMapping
    public ResponseEntity<?> updateProductInventory(@RequestBody
Inventory inventory) {
        // Update inventory directly in the database
        inventoryService.updateInventory(inventory);
        return ResponseEntity.ok().build();
    }

    // Other methods...
}
```

Compliant Code(Java):

```
@RestController
@RequestMapping("/api/inventory")
public class InventoryController {

    private final InventoryService inventoryService;

    public InventoryController(InventoryService inventoryService) {
        this.inventoryService = inventoryService;
    }

    // GET /api/inventory/{productId}
    @GetMapping("/{productId}")
    public ResponseEntity<Inventory> getProductInventory(@PathVariable
int productId) {
        // Fetch inventory data through the inventory service
        Inventory inventory =
inventoryService.getProductInventory(productId);

        if (inventory == null) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok(inventory);
    }

    // POST /api/inventory
    @PostMapping
    @PreAuthorize("hasRole('ADMIN')") // Restrict access to authorized
users with the "ADMIN" role
    public ResponseEntity<?> updateProductInventory(@RequestBody
Inventory inventory) {
        // Update inventory through the inventory service
        inventoryService.updateProductInventory(inventory);
        return ResponseEntity.ok().build();
    }

    // Other methods...
}
```

General Prevention Recommendations:

Comprehensive and precise documentation for the API, including current and previous versions.

Implementing a version management system that simplifies the updating and management of API versions.

Introducing a version release policy that includes the lifespan and support for outdated versions.

Using automated methods to check the API version used by clients and sending alerts if older versions are being used.

Continuous monitoring and auditing to detect and address issues like outdated API versions and faulty endpoints.

Employing automation techniques to examine and automatically update API versions and hosts.

Establishing update policies for outdated API versions and discontinuing support for them.

API10:2023 - Unsafe Consumption of APIs

Due to the aforementioned vulnerability, an attacker can execute their desired information or requests in a specific group by sending or receiving information from supply chain sources.

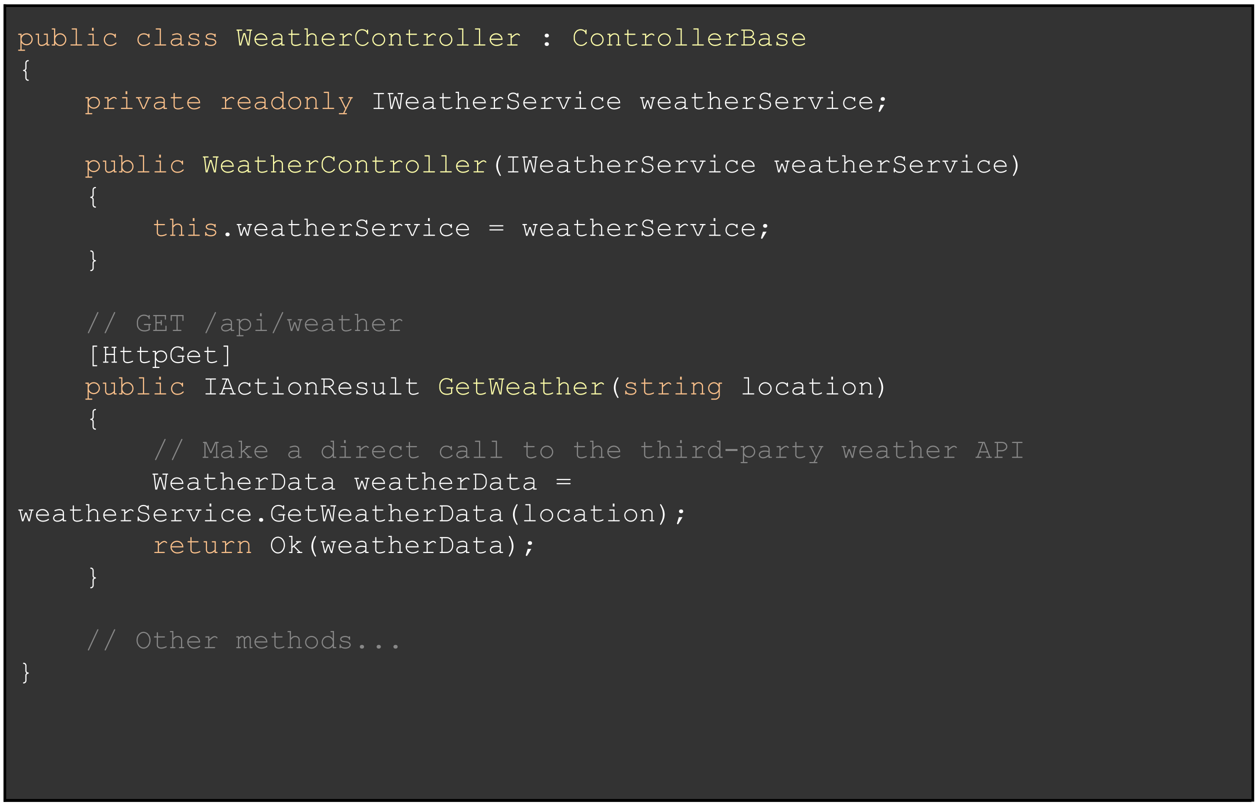
Example:

A GET request to retrieve weather information from a third-party service.

GET /api/weather?location=New+York

Noncompliant Code(.NET):

```
[ApiController]
[Route("api/weather")]
```



```
public class WeatherController : ControllerBase
{
    private readonly IWeatherService weatherService;

    public WeatherController(IWeatherService weatherService)
    {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    [HttpGet]
    public IActionResult GetWeather(string location)
    {
        // Make a direct call to the third-party weather API
        WeatherData weatherData =
weatherService.GetWeatherData(location);
        return Ok(weatherData);
    }

    // Other methods...
}
```

Compliant Code(.NET):

```
[ApiController]
[Route("api/weather")]
public class WeatherController : ControllerBase
{
    private readonly IWeatherService weatherService;

    public WeatherController(IWeatherService weatherService)
    {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    [HttpGet]
    public IActionResult GetWeather(string location)
    {
        // Validate the location parameter and restrict access to
trusted sources
        if (!IsValidLocation(location))
        {
            return BadRequest();
        }

        // Make a call to the third-party weather API through the
weather service
        WeatherData weatherData =
weatherService.GetWeatherData(location);

        if (weatherData == null)
        {
            return NotFound();
        }

        return Ok(weatherData);
    }

    private bool IsValidLocation(string location)
    {
        // Implement validation logic to ensure the location is safe and
trusted
        // This could involve white-listing trusted sources or
validating against a known set of safe locations
        // Return true if the location is valid, false otherwise
        // Example: return Regex.IsMatch(location,
"^[a-zA-Z]+(,[a-zA-Z]+)*$");
        // Implement your validation logic here

        // For simplicity, assuming any location is valid
        return true;
    }

    // Other methods...
}
```

Noncompliant Code(Java):

```
@RestController
@RequestMapping("/api/weather")
public class WeatherController {

    private final ThirdPartyWeatherService weatherService;

    public WeatherController(ThirdPartyWeatherService weatherService) {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    @GetMapping
    public ResponseEntity<WeatherData> getWeather(@RequestParam String
location) {
        // Make a direct call to the third-party weather API
        WeatherData weatherData =
weatherService.getWeatherData(location);
        return ResponseEntity.ok(weatherData);
    }

    // Other methods...
}
```

Compliant Code(Java):

```
@RestController
@RequestMapping("/api/weather")
public class WeatherController {

    private final ThirdPartyWeatherService weatherService;

    public WeatherController(ThirdPartyWeatherService weatherService) {
        this.weatherService = weatherService;
    }

    // GET /api/weather
    @GetMapping
    public ResponseEntity<WeatherData> getWeather(@RequestParam String
location) {
        // Validate the location parameter and restrict access to
trusted sources
        if (!isValidLocation(location)) {
            return ResponseEntity.badRequest().build();
        }

        // Make a call to the third-party weather API through the
weather service
        WeatherData weatherData =
weatherService.getWeatherData(location);

        if (weatherData == null) {
            return ResponseEntity.notFound().build();
        }

        return ResponseEntity.ok(weatherData);
    }

    private boolean isValidLocation(String location) {
        // Implement validation logic to ensure the location is safe and
trusted
        // This could involve white-listing trusted sources or
validating against a known set of safe locations
        // Return true if the location is valid, false otherwise
        // Example: return location.matches("^[a-zA-Z]+(,[a-zA-Z]+)*$");
        // Implement your validation logic here

        // For simplicity, assuming any location is valid
        return true;
    }

    // Other methods...
}
```

General prevention recommendations:

Approach data received from external APIs with caution and validate them thoroughly.

Investigate and ensure the security and security standards of the third-party service before connecting to it.

Use encryption when communicating with external services to prevent the transmission of sensitive information in plain text.

Limit access and set appropriate permissions for third-party services.

Implement protective mechanisms like sandboxing and generalization to ensure the security and reliability of data received from external services.

Continuously monitor and survey to detect and rectify any security flaws in external services.

Train developers on security principles and the proper use of external APIs.