🛡️ Introduction to OWASP API Security:
OWASP stands for Open Worldwide Application Security Project is a nonprofit organization dedicated to enhancing software security. Their creation of the OWASP Top 10, a list of the most important web application security threats, has earned them international recognition.
Since APIs face different threats than traditional web apps, OWASP released a Top 10 list specifically for APIs in 2019.
✅ The categories in the 2023 edition of the OWASP API Security Top 10 were improved to take into account fresh threats, actual data, and new tactics used by attackers.

## 🧩 OWASP API Top 10 – #1: Broken Object Level Authorization (BOLA):

Definition 🧠
When an API improperly verifies whether a user is authorized to access or modify a particular object (such as user data, accounts, or assets), BOLA happens. The way access control is implemented at the object level is illogical.

### 📣 The Reason It's #1 (Again in 2023)

It is dangerous, common, and difficult to identify.
Attackers can easily guess or alter objects referenced by user-controllable values (e.g., /api/users/123) in API requests.
These kinds of logical errors are rarely detected by conventional security tools (WAFs, DAST).

### 🧪 Real-World Instances 🔶

1. Coinbase: Trade Without Asset Ownership
   What transpired was that a hacker converted a legitimate Ethereum trade to Bitcoin, which they did not possess.
   Error message is the anticipated outcome.
   Real outcome: The trade was successful. $43,000 in Bitcoin was converted from $1,060 in Ethereum.
   Root cause: The API neglected to perform a logic check to confirm that the asset truly belonged to the user.
   Impact: The possibility of disastrous market manipulation.
   Important takeaway: The product_id (asset) being sold was not validated by the API. The API didn't block it, but the user interface did.

2. Peloton: Exposure of 4 Million User Records
   What transpired: Without requiring credentials, an unprotected API endpoint revealed all user data, including profile information.
   Added authentication as the first fix.
   Issue: The data of other users was still accessible to any logged-in user.
   Root cause: The API failed to verify who the user was authorized to access because authorization checks were absent.
   Important lesson: Authorization ≠ Authentication Being aware of "who you are" does not imply that "you should have access to this dat."

## 🔐 OWASP API Top 10 – #2: Broken Authentication

📖 Definition
Unauthorized users can access protected resources or carry out sensitive operations when an

API has weak, missing, or malfunctioning authentication mechanisms. This is known as broken authentication.

## ⚠️How It Appearances:

No authentication whatsoever (open APIs)
Weak mechanisms (such as the absence of CAPTCHA and 2FA)
Session management issues (hardcoded tokens, non-expiring tokens, etc.)
Brute-force vulnerability and credential stuffing
Permitting enumeration, such as "Is this email real?"

## 📉Real-World Example: Duolingo Breach

App: Duolingo, a platform for learning languages
Incident: An unauthenticated API exposed more than 2 million user profiles.
How It Occurred:
Anyone with an email address could query user profiles thanks to the API.
The API provided comprehensive information in response, including the user's learning language, profile picture, and streak.
It's totally open with no rate limitation, authentication, or detection.
Effect:
Exposure of PII (user emails, profile information)
could be applied to specific doxing, scraping, or phishing campaigns.
🔍 With just a browser, even a non-technical user could take advantage of it!

## 🧠 Authentication vs. Authorization

Verify your identity through authentication (e.g., login, token).
Authorization: Establish your access restrictions (e.g., only your own data).
Duolingo's authentication failed. Peloton did not have permission. Security failed for both.

## 🔓 Why Do APIs Remain Open?

Developers are under pressure to release features as soon as possible.
"It is merely an internal API."
"No one is aware of this endpoint."
"Managing tokens during development is too difficult."
The truth is that attackers are searching for these unnoticed endpoints.

## ✅ The Best Ways to Avoid Authentication Issues:

1. By default, secure all endpoints
   No unauthenticated APIs unless specifically permitted
   For all production APIs, use token-based authentication (such as JWT or OAuth2).
2. Make use of MFA, or multi-factor authentication.
   Particularly for high-risk activities (like changing passwords or transferring money).
3. Put CAPTCHA and rate limitation into practice.
   Avoid credential stuffing and brute-force attacks.
   To identify questionable activity, use behavioral detection.
4. Security of Tokens
   Strongly validated refresh tokens and temporary access tokens
   Tokens are invalid when you log out or change your password.

5. Steer clear of user listing
   Don't give "user not found" and "wrong password" different errors.
   Avoid using queryable APIs to reveal user existence.
6. Ongoing Examination
   Auth enforcement should be validated by automated security tests.
   Incorporate unauthorized and unauthenticated access attempts into test cases.

## 🧰 Tips for Developers:

To test endpoints without the user interface, use programs like Postman or Burp Suite.
Put OpenAPI specifications and contract-based tests for authentication rules into practice.
Make sure roles and scopes are enforced server-side by utilizing JWT claim-based routing.

## 🧩 OWASP API Top 10 – #3: Broken Object Property Level Authorization (BOPLA)

📖 Definition

BOPLA happens when an API:
   exposes more information than is required (previously known as excessive data exposure), and/or
   permits changing object properties that a user shouldn't be able to alter (previously Mass Assignment)
   It is therefore a dual-risk category with a fine-grained authorization focus.

## 🧠 Understanding BOPLA Through Examples

🔹 Overexposure to Data (Read)
   Too much information is returned by APIs, including fields that ought to be hidden.
   Example: in a mobile application where only the user's first name is shown, returning the user's SSN, home address, or bank account number.

🔹 Write a mass assignment
   Fields that users shouldn't be able to alter are modified by APIs.
   Example: When requesting a profile update, a normal user sets their role to "admin."

## 📉 Real-World Example: Harvesting Venmo Transactions

Venmo is a payment app that has a social media feed.
Problem: Anonymized transactions (such as "John paid Sarah 🍕") were displayed on the homepage.
What Went Wrong:
Complete transaction records were returned by the public API that powered that feed:
Even when the user interface only displayed first names, complete names, comments, and timestamps were displayed.
A researcher harvested more than 200 million complete records by scripting API calls.
The underlying cause
The UI was used to "hide" fields in the API, which was built to return complete records.
However, attackers target the raw API directly rather than through the user interface.

## ✅ Top Tips for Avoiding BOPLA 🔒 For Data Exposure (Read):

Minimization of Data
Only return fields that are specifically needed by the business logic.
For instance, the API should only return the name and status if that is all that is shown in the

user interface.
Distinct Private and Public APIs
Make specific, limited versions of APIs for use by the general public.
Steer clear of externally reusing internal APIs.
Make use of serialization controls and response filtering.
Make use of libraries that mandate that response fields be whitelisted.

## 🛡️ For Writing Mass Assignments:

Clearly Define the Writable Fields
Complete request bodies should never be bound directly to database objects (e.g., User.from(request.body)).
Make use of "property whitelists," also known as field-level allow lists.
Implement Role Checks on the Server Side
Even if users include sensitive fields (like roles or permissions) in their request, don't let them change them.
Make use of schema validation
Use tools such as these to apply stringent validation to input fields:
Schema in JSON
Joi (Node.js)
Python-based Pydantic

## 🧪 Strategies for Testing

Utilize tools for API scanning to find:
API responses that contain sensitive information (PII, credentials, tokens)
Properties that can be written but shouldn't be changed
Designing for Security:
During design, rather than after development, specify writable fields and API response fields.

## 🧱 OWASP API Top 10 – #4: Unrestricted Resource Consumption

📖 Definition
APIs that don't restrict the use of server-side resources are known as "unrestricted resource consumption," which allows:
Attacks using brute force and credential stuffing
Large-scale data scraping
DoS (Denial of Service) and server overloads
OWASP 2019 referred to this risk as "Lack of Resources & Rate Limiting," but it has since expanded to encompass execution time, memory, response sizes and backend operation abuse.

## 🔁 Real-World Example: Trello API Abuse

App: Atlassian's Trello
Feature: Send an email inviting a user to a project
The invite API endpoint was vulnerable.
Open to the public (no authentication needed)
returned comprehensive user information when an email was sent.
What the assailant did:
500 million compromised email addresses were obtained.

500 million API requests were sent.
collected user profiles through numerous, anonymous API requests
Methods employed:
Home-based proxies
Dispersed IPs
Requests should be made gradually to avoid detection thresholds.

## 💣 Impact of Unrestricted Consumption
Large-scale data scraping
Deterioration of system performance
excessive use of backend resources (such as database hits)
Bypassing security infrastructure (e.g., WAF, bot detection)

## ✅ Best Practices to Prevent Resource Abuse
1. Implement Rate Limits for Users and IPs
Make use of leaky bucket algorithms or sliding window counters.
Increase the restrictions for users who are not authenticated.
2. Make Sensitive Actions Require Authentication
For high-value or profile-specific endpoints, tokens should be required even for public APIs.
3. Put Resource Constraints in Place
Timeouts: Limit how long an API call can take to execute.
5. Put Your Defenses to the Test
In staging, mimic physical force and volumetric abuse.
Make sure your WAF/API gateway is operational and not just a checkbox.
Maximum Payload Size: Restrict JSON payloads or file uploads
Never return large datasets without controls using pagination.
4. Track, Identify, and Prevent Abuse
Record any unusual increases in traffic for each user or IP.
Don't rely solely on static thresholds; use behavioral threat detection.

## 🧮 OWASP API Top 10 – #5: Broken Function Level Authorization (BFLA)
📖 Definition
When APIs don't appropriately limit what a user can do, it's known as Broken Function Level Authorization (BFLA).
🔄 The "action-based twin" of OWASP #1 (BOLA), which deals with data access, is BFLA, which deals with action permissions.

## 📉 Real-World Case Study: Bumble Account Upgrade Hack
App: The dating app Bumble
User Roles: Premium vs. Free Accounts .
API Defect:
Users can view their subscription level via a GET endpoint.
However, users could modify their subscription status using a POST method on the same endpoint.
As a result, users could upgrade to premium without having to pay.
The underlying cause
Without appropriate role or privilege checks, the upgrade function was made publicly

available.
The backend failed to confirm that a payment had been made.

## 🛑 Dangers of Invalid Function Permission

Escalation of privileges (free to premium, user to administrator)
Unauthorized actions (deleting, resetting, altering data belonging to others)
Financial loss and fraud
Tampering or taking over an account

## 🔒 Best Practices to Prevent BFLA

1. Put Role-Based Access Control (RBAC) into practice.
   Specify which actions are permitted for each role (user, admin, and guest).
   Implement authorization logic not only in the frontend but also at the API layer.
2. Employ Role-Specific Methods or Distinct Endpoints
   Separate write (POST/PUT/DELETE) and read (GET) operations
   Never grant users indirect access to admin-only endpoints.
3. Don't Depend on the User Interface to Restrict Actions
   Although frontend buttons may be hidden, APIs are not.
   Attackers make direct API calls using programs like Postman, Burp Suite, or curl.
4. Check Every Function Against Every Role
   Make sure that unauthorized users are unable to carry out sensitive or admin-level tasks.
   Create test cases that replicate:
   Visitors attempting to remove data
   Attempts by free users to switch account types
   Ordinary users using admin-only functions