radware

White Paper

Understanding the OWASP API Top Ten for 2023 (and How to Protect Your APIs)

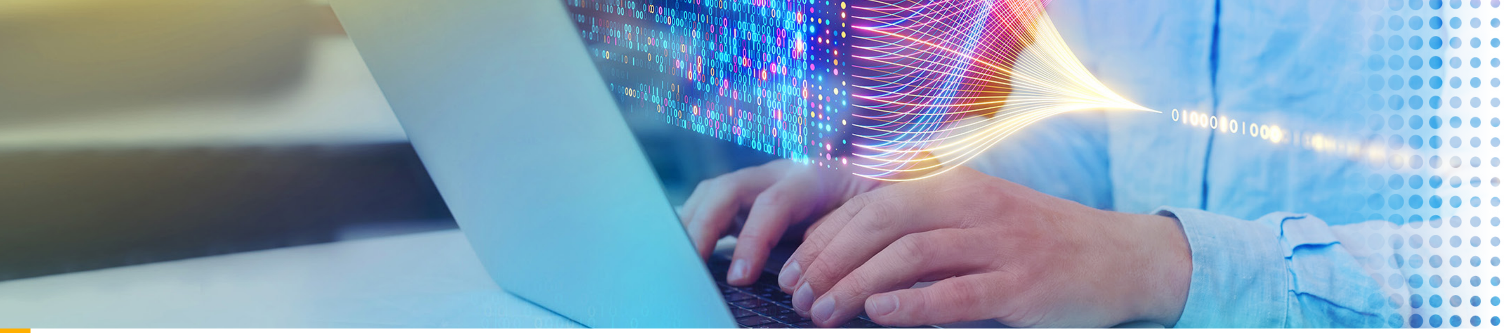

Table of Contents

In today's interconnected digital landscape, application programming interfaces (APIs) play a critical role in facilitating communication between software applications. However, with this convenience comes the responsibility of ensuring API security to protect against malicious attacks. Let's delve into key aspects of API security using insights from Open Web Application Security Project (OWASP) API Security Top Ten for 2023. Let's explore the fundamental concepts of API security and learn how Radware helps protect your APIs against each of those vulnerabilities.

OWASP API 2023 – Risk 1 : Broken Object-level Authorization

This risk occurs when APIs expose endpoints that manage object identifiers, creating a wide attack surface of Object-level Access Control issues. Object-level authorization checks should be considered for every object of a function or endpoint accessed by a specific user.

Object-level authorization is an access control mechanism that is usually implemented at the code level to validate that a user can only access the objects they should have permission to access.

Example Scenario: Profile Access

Consider an application that allows users to view their own profile information via an API call:

GET /api/profile/{user_id}

An attacker, instead of using their own user_id, substitutes it with the user_id of another user:

GET /api/profile/other_user_id

If the API lacks proper authorization checks, the attacker gains unauthorized access to another user's profile information.

OWASP 2023 API – Risk 2: Broken Authentication

This risk occurs when authentication mechanisms are implemented incorrectly, allowing attackers to compromise authentication tokens or exploit implementation flaws to assume other users' identities temporarily or permanently. Compromising a system's ability to identify the client/user compromises API security overall.

Example Scenario 1: Money Transfers

Imagine an online banking application that uses a token-based authentication system. The API endpoint for transferring funds looks like this:

```
POST /api/transfer_funds
Authorization: bearer “valid_token_here”
{
   “recipient_account”: “12345678”,
   “amount”: 100.00,
   }
```

After an attacker discovers a valid token (e.g., by stealing it from a legitimate user's device or intercepting it during transmission), the attacker crafts a malicious request:

```
POST /api/transfer_funds
Authorization: bearer “valid_token_here”
{
   “recipient_account”: “attacker_account”,
   “amount”: 1000000.00,
 }
```

Due to improper authentication checks, the API processes the request, transferring a large amount of money to the attacker's account.

Example Scenario 2: Credential Stuffing

Credential stuffing involves attackers using lists of compromised username/password pairs, often obtained from previous data breaches, to gain unauthorized access to user accounts. Here's a detailed example of how an API that's vulnerable to broken authentication might be exploited via credential stuffing:

An API for an online retail service allows users to log in to their accounts. The authentication endpoint does not have sufficient protection against automated login attempts.

```
Endpoint:
POST /api/login
Payload: {“username”: “user@example.com”, “password”: “password123”}
```

One can use a simple python script to automate login with a list of user names / passwords:

```
import requests
url = "https://api.onlineretail.com/api/login"
# List of compromised credentials
credentials = [
    {"username": "victim1@example.com", "password": "password123"},
    {"username": "victim2@example.com", "password": "123456"},
    # More credentials...
]
for cred in credentials:
    response = requests.post(url, json=cred)
    if response.status_code == 200 and "token" in response.json():
        print(f"Successful login: {cred['username']}")
        # Store or use the token for further actions
    else:
        print(f"Failed login: {cred['username']}")
```

OWASP API 2023 – Risk 3: Broken Object Property Level Authorization

This category combines two vulnerabilities: Excessive Data Exposure and Mass Assignment. The root cause lies in the lack of or improper authorization validation at the object property level. APIs vulnerable to this risk expose object properties without adequate authorization checks. Essentially, it's about controlling access to specific properties within an object.

Example Scenario: Information Changes

Consider an e-commerce application with an API endpoint that retrieves product details:

GET /api/products/{product_id}

The API returns the entire product object, including sensitive properties like price, cost, and supplier details. An attacker can directly call this API and change sensitive information that should be restricted. For instance: PUT /api/products/123

The request includes:

```
{
    "product_id": 123,
    "name": "Super Secret Widget",
    "price": 0,          ← this could be a manipulation of the attacker, reducing the price
    "cost": 100.00,
    "supplier": "Confidential Corp"
    ...
}
```

Proper authorization checks at the property level would prevent unauthorized write/read access to sensitive fields that should only be accessible by administrators.

OWASP API 2023 – Risk 4 : Unrestricted Resource Consumption

This vulnerability occurs when APIs do not properly limit client interactions or resource consumption. It is like managing a buffet where guests can take as much food as they want. Here is a non-technical breakdown:

Resource Hunger: APIs are hungry for resources like bandwidth, CPU, memory, and storage. For example, attackers can bombard APIs with requests, gobbling up resources.

Denial of Service (DoS): Successful attacks can starve the system, causing a Denial of Service. It is like a buffet that runs out of food because one person ate everything.

Example Scenario 1: Overwhelming Requests

A social network's "forgot password" flow uses SMS verification. An attacker sends thousands of API requests, causing the system to send tens of thousands of text messages, resulting in significant financial losses.

Example Scenario 2: Large Image Upload

A GraphQL API allows users to upload profile pictures. Due to insufficient resource limits, an attacker uploads a large image, exhausting available memory and rendering the API unresponsive.

OWASP API 2023 – Risk 5: Broken Function Level Authorization

Imagine a building with various rooms, each requiring different keys for access. If those rules are unclear or misconfigured, intruders might gain access to restricted areas. Similarly, in APIs, flawed authorization can allow attackers to access resources they should not, potentially compromising sensitive data or functions. Exploitation of this risk requires the attacker to send legitimate API calls to an endpoint they should not have access to (e.g., as anonymous users or regular non-privileged users). These exposed endpoints become easy targets for exploitation. Authorization checks for a function or resource are typically managed via configuration or code level.

Example Scenario: Registration Duplication

Consider an application that allows only invited users to join. During the registration process, the mobile app triggers an API call to retrieve details about an invite using *GET /api/invites/{invite_guid}*. The response includes the user's role and email.

An attacker duplicates the request, manipulates the HTTP method and sends a malicious request to *POST /api/invites/new*. This endpoint should be accessible only by administrators via the admin console. Unfortunately, the endpoint lacks proper function level authorization checks. The attacker exploits this flaw by creating a new invite with admin privileges:

```
POST /api/invites/new
{
   "email": "attacker@somehost.com",
   "role": "admin"
}
```

OWASP API 2023 – Risk 6 : Unrestricted Access to Sensitive Business Flows

This vulnerability occurs when APIs do not properly limit client interactions or resource APIs vulnerable to this risk expose a business flow—such as buying a ticket or posting a comment—without adequately considering how the functionality could impact the business if abused in an automated manner.

Example Scenario 1: Manipulating Financial Transactions

An API might expose endpoints that handle financial transactions, such as transferring money between accounts. If access controls are not properly implemented, an attacker could:

- Access endpoints meant for administrators or high-privilege users.
- Change transaction details (e.g., amount, recipient) without proper authorization.
- Trigger unauthorized transactions.

Example Attack:

Endpoint: *POST /api/transfer*

*Payload: {"fromAccount": "123", "toAccount": "456", "amount": "1000"}*If the API lacks proper access control checks, any authenticated user might be able to execute this endpoint, transferring funds without authorization.

Example Scenario 2: Changing Account Permissions

APIs that manage user roles and permissions should be strictly controlled. An attacker could exploit this vulnerability to escalate their privileges.

Example Attack:

Endpoint: *POST /api/users/changeRole*

Payload: {"userId": "789", "role": "admin"}

Without proper access control, an attacker with basic user privileges could change their role to an administrator.

OWASP API 2023 – Risk 7: Server-Side Request Forgery (SSRF)

SSRF flaws occur when an API fetches a remote resource without validating the user-supplied URI. Attackers can coerce the application to send crafted requests to unexpected destinations, even bypassing firewalls or VPNs. It is like a sneaky trick where an attacker manipulates an authorized application into fetching restricted information on their behalf.

Example Scenario: Profile Picture Upload API

Imagine a social network that allows users to upload profile pictures. When a user chooses to provide an image URL instead of uploading a file from their machine, the following API call is triggered:

```
POST /api/profile/upload_picture

{
   “picture_url”: “http://example.com/profile_pic.jpg”
}
```

An attacker can exploit this by sending a malicious URL like “localhost:8080”. The API endpoint will attempt to fetch the resource from that URL. Depending on the response time, the attacker can determine whether the port is open or not within the internal network.

OWASP API 2023 – Risk 8: Security Misconfiguration

This risk occurs when the security configuration settings of an API are not properly set up. It’s like accidentally sharing your secret codes with everyone. Think of it as leaving your house unlocked or your safe combination written on a sticky note. Attackers can find unpatched flaws, unprotected files, or common endpoints, gaining unauthorized access or knowledge of the system.

Example Scenario: Finding Hidden Endpoints

An attacker discovers an undocumented API endpoint used only by the DevOps team. They send a request to this endpoint, accessing sensitive functions or data. Without proper security configuration, the system unwittingly spills its secrets.

OWASP API – Risk 9: Improper Inventory Management

This risk emphasizes the importance of maintaining accurate documentation and tracking deployed API versions to mitigate vulnerabilities associated with exposed endpoints and deprecated versions.

Example Scenario: Outdated endpoints

Consider an API that lacks proper inventory management. If deprecated endpoints or outdated versions remain accessible, an attacker could exploit them to gain unauthorized access or manipulate sensitive data.

OWASP API 2023 - 10: Unsafe Consumption of APIs

This risk highlights the tendency of developers to trust data received from third-party APIs more than user input. As a result, they may adopt weaker security standards. Instead of directly compromising the target API, attackers often go after integrated third-party services to achieve their malicious goals.

Example Scenario: Gateway Vulnerabilities

Developers tend to trust data received from third-party APIs more than user input, leading to weaker security practices. Attackers exploit integrated third-party services to compromise APIs indirectly. Imagine an e-commerce application that integrates with a payment gateway API. If the payment gateway API has vulnerabilities, attackers might abuse it to manipulate transactions, steal funds or gain unauthorized access to sensitive data.

Effective Protection Against the OWASP Top 10 Vulnerabilities

Effective protection of APIs can't be applied with a single tool, just like API attacks are not executed through a single API call. Sophisticated API attacks often include a scanning phase, a search for vulnerabilities, the learning of the API endpoints, and many other techniques to prepare attacks that are often executed through innocent looking API calls. To block even the most sophisticated type of attacks, one would need a combination of API protection tools to detect the attack as soon as possible (e.g. in the scanning phase, or better yet, blocking known bad actors), and then apply accurate blocking with surgical precision, so that it won't cause false positives and break the application.

Understanding the Tools That Protect Your APIs

Token-Based Authentication: Ensures only authenticated and authorized users interact with your APIs.

Radware's token-based system safeguards sensitive data and prevents unauthorized access and data tampering, effectively addressing various OWASP API Top 10 concerns.

Client source access list: Enables strict control over API access by allowing administrators to specify authorized IP addresses and geographical locations, effectively blocking malicious attempts from illegitimate sources.

Radware enforces these access restrictions to ensure that only requests originating from trusted sources are allowed, thereby mitigating the risk of unauthorized access and thwarting potential attacks outlined in the OWASP API Top 10. This granular control mechanism enhances security posture by preventing unauthorized entities from interacting with the API endpoints, strengthening the overall defense against threats such as unauthorized access and potential data breaches.

Automatic API Discovery: A must-have in a reality where APIs are changing on daily basis and often be undocumented or from third parties.

Radware stands out with its ability to automatically learn all API endpoints and their structure and automatically produce a detailed and accurate schema file, which can enable accurate API protection.

Our API Discovery goes on to provide regex for enforcement, a level of protection that surpasses standard developer tools. It uncovers hidden vulnerabilities like zombie APIs and outdated endpoints, empowering you to fortify your API infrastructure with confidence. Boost security, build trust and streamline API interactions effortlessly with our straightforward API Discovery solution.

API Schema Enforcement: Rigorously upholds the integrity of traffic structure and format, safeguarding against malicious manipulations in both REST and GraphQL APIs.

By strictly enforcing predefined schemas, Radware ensures that all incoming API calls adhere to expected formats, mitigating the risk of exploitation and defending against potential threats outlined in the OWASP API Top 10. This proactive approach bolsters the security posture of APIs.

Quota Enforcement Per API Endpoint: A pivotal feature designed to protect against vulnerabilities and prevent excessive data consumption and manipulation.

By imposing a rate limit on each API Endpoint, Radware not only deters potential attackers from exploiting vulnerabilities through repeated requests, but also shields your system from malicious actors attempting to uncover data structures for more potent attacks and potential data leakage. This proactive measure ensures that your organization remains safeguarded against a wide range of threats, bolstering your overall security posture and providing peace of mind for your team.

Brute Force Defense: An advanced safeguard against unauthorized access attempts.

Radware implements robust measures to defend against malicious users attempting to gain entry through repetitive login attempts. By enforcing stringent login protection, this protection mechanism shields your platform from brute force attacks, ensuring malicious actors won't be able to guess credentials that grant access to your APIs and applications.

Bad Bot Protection: Vital for safeguarding against Bots and capable of emulating human behavior or legitimate actors.

Radware's Bot Manager solution effectively distinguishes between legitimate users and malicious bot activity through various techniques that eliminate many protection avoidance tricks used by advanced bots.

Data Leakage Prevention (DLP): A crucial feature for safeguarding sensitive data such as personal identifiable information (PII), credit cards and social security numbers from leaking outside your applications through legitimate looking API calls.

By inspecting all server responses, and detecting PIIs within them, Radware masks the sensitive data in the API call response, protecting it from leaking outside.

Business Logic Protection – Sequencing: A powerful protection tool that can outsmart evolving threats that try to access and manipulate data by leveraging vulnerabilities in the API flow (the business logic).

Radware's API Protection solution continuously learns legitimate API call flows, and thus it can detect and block in real-time non-legitimate call flaws before they exert any harm to the application or its data integrity.

GraphQL Protection: A protection designed to mitigate risks associated with excessive data manipulation, exposure and server resource consumption, while also safeguarding against malicious actors collecting more information than necessary.

Radware's API Protection solution enforces strong input validations with the option to strengthen strict limits within the GraphQL language, preventing potential abuse that could compromise data integrity and strain server resources. By imposing this protection, we ensure that only necessary and authorized data requests are processed, while ensuring optimal performance of GraphQL servers.

ActorID Security Enforcement: IP-based mitigation of bad actors is no longer effective or enough, as API based attacks can often bypass this type of mitigation or cause an unacceptable level of false positives.

Radware's API Protection solution prioritizes security at the level of ActorID, aligning seamlessly with the intricacies of your application's business logic flows, ensuring granular protection tailored to each unique ActorID (i.e., malicious user/actor).

Client-Side Protection: Encompasses various measures to secure the user's end of interactions with applications. This includes safeguarding sensitive data stored locally, such as in local storage, from unauthorized access or tampering. Additionally, mitigation against threats like "man-in-the-browser" attacks ensures that malicious entities cannot intercept or manipulate data exchanged between the client and server or client and malicious server.

Furthermore, client-side login mechanisms, coupled with protection of personally identifiable information (PII), help fortify authentication processes and shield sensitive user data from unauthorized access.

Preventing token leakage ensures that authentication tokens or session identifiers are not inadvertently exposed, reducing the risk of unauthorized access to user accounts or sensitive resources. These collective efforts in client-side protection bolster overall security posture, instilling confidence in users and safeguarding against a myriad of potential threats.

Tailored Protection for Each OWASP API Vulnerability

The following table specifies which combination of protection tools are needed for each OWASP API vulnerability.

	OWASP API 1	OWASP API 2	OWASP API 3	OWASP API 4	OWASP API 5	OWASP API 6	OWASP API 7	OWASP API 8	OWASP API 9	OWASP API 10
Token Control Authorization	✓	✓	✓		✓	✓		✓		
Client Source Access List	✓	✓	✓	✓	✓	✓		✓		✓
Automatic API Discovery	✓		✓	✓	✓		✓	✓	✓	✓
API Schema Enforcement	✓		✓	✓	✓		✓	✓	✓	✓
Quota Enforcement Per API Endpoint	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Credential Cracking (Brute Force Protection - Part of ATO)		✓		✓		✓		✓	✓	
Bad Bot Protection		✓		✓		✓		✓		
Data Leakage Prevention (DLP)	✓		✓		✓			✓		✓
Business Logic Protection – Sequencing	✓	✓	✓	✓	✓	✓	✓	✓	✓	
GraphQL Protection			✓	✓				✓	✓	
ActorID Security Enforcement	✓	✓	✓	✓	✓	✓		✓		
Client-Side Protection								✓		✓
Signature-based Attack Protection				✓			✓	✓		✓

Summary

APIs represent the broadest attack surface in enterprise applications, using diverse attack vectors ranging from SQL injection to complex manipulations of legitimate API calls to bypass business logic. OWASP's 2023 guidelines highlight common API vulnerabilities, such as broken object-level authorization and excessive data exposure, necessitating a comprehensive protection approach.

Effective API security requires a combination of tools to detect attacks, enforce accurate security policies, and block malicious actors while ensuring legitimate users are not affected. A good API protection solution should include automatic discovery API endpoints and their structure, auto-learning of the application's business logic, real-time attack detection (based on the above), and mitigation with minimal false positives.

Radware's API Protection solution leverages AI-based algorithms to continuously learn and adapt to API structures and behaviors, translating this knowledge into precise security policies that protect against evolving threats while maintaining service continuity for legitimate users.