OWASP API Security Top 10

OWASP (Open Web Application Security Project) publishes a list of the top 10 security risks associated with web applications, including those related to APIs (Application Programming Interfaces). Here are the **OWASP API Security Top 10** vulnerabilities, explained:

1. Broken Object Level Authorization (BOLA)

- **Description**: Occurs when an API endpoint does not properly check if a user is authorized to access or manipulate a specific object, like user data.
- **Impact**: Attackers can access or modify data they should not have access to by manipulating object IDs.
- **Example**: A user accessing another user's profile data by changing a user ID in the request.

2. Broken Authentication

- **Description**: Refers to improper implementation of authentication mechanisms, allowing attackers to impersonate other users.
- **Impact**: Attackers may be able to gain access to user accounts, often due to weak passwords, session management flaws, or broken token validation.
- **Example**: Use of easily guessable or non-expiring tokens in an API.

3. Excessive Data Exposure

- **Description**: Occurs when APIs return more data than necessary, relying on the client to filter the information.
- **Impact**: Sensitive data that should remain hidden can be exposed, even if it is not used by the client-side application.
- **Example**: An API response returning personal details (such as Social Security numbers) along with non-sensitive data.

4. Lack of Resources & Rate Limiting

- **Description**: When an API does not enforce rate limiting, attackers can overload the server by sending too many requests.
- **Impact**: This can lead to Denial of Service (DoS) attacks or brute-force attacks on authentication endpoints.
- **Example**: A login endpoint that doesn't limit the number of login attempts, allowing brute force attacks.

5. Broken Function Level Authorization

- **Description**: This vulnerability occurs when the API doesn't properly enforce authorization checks at the function level, allowing users to perform unauthorized actions.

- **Impact**: Attackers could gain access to privileged functions or data they should not have access to.
- **Example**: A regular user performing admin-level actions by manipulating requests.

6. Mass Assignment

- **Description**: Happens when an API automatically binds client-supplied data to internal object properties without proper filtering.
- **Impact**: Attackers can provide unexpected fields in requests and update sensitive properties such as roles or permissions.
- **Example**: A user updating their account information can also update their role to ‘admin’ due to improper request validation.

7. Security Misconfiguration

- **Description**: Refers to the incorrect setup or configuration of API components, such as servers, frameworks, or security settings.
- **Impact**: Misconfigured settings (like leaving default credentials or enabling unnecessary features) can expose the API to attacks.
- **Example**: Using default settings for databases or not disabling debugging features in production environments.

8. Injection Attacks

- **Description**: Occurs when user input is not properly sanitized, allowing attackers to inject malicious code into the API, often leading to data leaks or server control.
- **Impact**: Attackers can execute commands, retrieve sensitive data, or modify the database.
- **Example**: SQL Injection, where an attacker sends SQL queries through API parameters to manipulate the database.

9. Improper Assets Management

- **Description**: Happens when organizations fail to manage API versions and environments, such as leaving outdated APIs publicly accessible.
- **Impact**: Attackers can exploit old or undocumented APIs with security flaws that are no longer maintained.
- **Example**: An old API version still accessible, exposing sensitive vulnerabilities that have been fixed in newer versions.

10. Insufficient Logging & Monitoring

- **Description**: Failing to implement proper logging and monitoring can lead to delayed detection of attacks or suspicious activity.
- **Impact**: Without timely detection, attacks can go unnoticed for long periods, increasing the damage done.
- **Example**: A brute-force attack on login credentials going undetected due to a lack of monitoring and alerting.

Summary Table:

Rank	Vulnerability	Description & Impact
1	Broken Object Level Authorization	Unauthorized data access by manipulating object IDs
2	Broken Authentication	Attackers impersonate users due to weak authentication
3	Excessive Data Exposure	More data exposed than necessary, leaking sensitive information
4	Lack of Resources & Rate Limiting	Overloading the server or brute force attacks on login
5	Broken Function Level Authorization	Unauthorized function access by exploiting role mismanagement
6	Mass Assignment	Attacker modifies object properties like roles by sending unexpected fields
7	Security Misconfiguration	Default settings, unnecessary features left enabled
8	Injection Attacks	Malicious code injected via unsanitized input (e.g., SQL Injection)
9	Improper Assets Management	Outdated or insecure API versions still accessible
10	Insufficient Logging & Monitoring	Delayed detection of attacks due to poor monitoring

Understanding and mitigating these vulnerabilities is essential to securing APIs and protecting both applications and users from potential threats.

API Security Measures

1. **HTTPS (SSL/TLS Encryption)**: Ensure all API traffic is encrypted using HTTPS to protect data in transit.
2. **OAuth 2.0**: Use OAuth 2.0 for authorization, providing secure access without exposing user credentials.
3. **WebAuthn (Web Authentication)**: Implement WebAuthn for strong, passwordless authentication for web applications.
4. **Leveled API Keys**: Use API keys with different access levels (e.g., read-only, admin) to ensure proper access control.
5. **Authorization (Role-Based Access Control)**: Implement fine-grained access controls to ensure users or services have only the necessary permissions.
6. **API Versioning**: Implement versioning to allow safe deprecation of old APIs and maintain backward compatibility.
7. **IP Allowlisting/Whitelisting**: Limit access to the API to specific IP addresses or ranges to reduce exposure to unwanted traffic.
8. **OWASP API Security**: Regularly check and adhere to OWASP API Security best practices to mitigate top API threats (e.g., injection, broken authentication).
9. **API Gateway**: Use an API gateway for traffic management, rate limiting, and security enforcement to protect APIs from common threats.
10. **Error Handling**: Avoid exposing sensitive details in error messages. Provide generic error responses that do not reveal internal mechanisms.
11. **Input Validation and Sanitization**: Ensure all inputs are properly validated and sanitized to prevent SQL injection, XSS, and other common attacks.
12. **Authentication**: Require strong authentication mechanisms (e.g., OAuth 2.0, API keys, JWTs) to ensure only authorized users access the API.
13. **Authorization**: Implement role-based access controls to ensure users have the correct permissions for each resource.
14. **Output Encoding**: Encode outputs to prevent the introduction of executable code and mitigate risks like XSS attacks.
15. **Logging and Monitoring**: Implement logging of all API interactions, including authentication failures and suspicious activities, and monitor these logs regularly for signs of attacks.
16. **Rate Limiting**: Control the number of requests an API client can make to prevent abuse and mitigate denial-of-service (DoS) attacks.
17. **Token Expiry**: Ensure that tokens (like JWTs) have a short lifespan and require re-authentication to limit exposure from compromised tokens.
18. **CORS (Cross-Origin Resource Sharing)**: Configure CORS policies to control which domains can access the API.
19. **Data Encryption (At Rest and In Transit)**: Encrypt sensitive data at rest and in transit using appropriate encryption standards (e.g., AES, TLS).
20. **Cross-Site Request Forgery (CSRF) Protection**: Use anti-CSRF tokens to ensure that requests are legitimate and not from forged sources.
21. **API Throttling**: Implement throttling to prevent API abuse and protect against large-scale attacks like brute-force attempts.
22. **Security Headers**: Use security headers such as `X-Content-Type-Options`, `X-Frame-Options`, and `Content-Security-Policy` to mitigate attacks.
23. **Traffic Filtering (DDoS Protection)**: Use firewalls, intrusion detection systems (IDS), and API gateways to filter traffic and block malicious requests.
24. **HMAC (Hash-based Message Authentication Code)**: Use HMAC for verifying message integrity and ensuring data authenticity.
25. **Session Management**: Implement secure session management techniques, including session timeouts and revocation.
26. **Egress Filtering**: Control outbound API traffic to prevent data leakage or communication with unauthorized third parties.
27. **Security Audits and Penetration Testing**: Conduct regular security audits and vulnerability assessments to identify weaknesses in the API.
28. **API Documentation Security**: Secure API documentation portals with authentication and restrict access to sensitive internal documentation.