wallarm

API Security Checklist

The practical guide to secure your APIs

Not sure where you stand with API security? This checklist is for you. We share common API security issues, their implications, and mitigation strategies. The checklist can serve as a starting point for Engineering and Security teams looking to keep APIs compliant and secure.

2022

wallarm

Intro

APIs come in many flavors, including REST, SOAP, graphQL, gRPC, and WebSockets, and each has its own use cases and common vulnerabilities. The issues covered in this checklist can occur in any kind of API. Regardless of which technology you have used to implement your API, read on to find out what you can do today to address the biggest potential risks associated with it.

API Checklist

wallarm

Improper API asset management and discovery

APIs present a large attack surface as each action requires its own endpoint. Ensuring all endpoints are identified and fully documented allows for potential attack vectors to be recognized and mitigated or monitored.

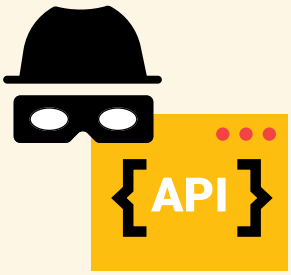
Keep track of your assets and infrastructure to ensure they are properly secured

- **Make an inventory of all API infrastructure** (from testing to production), including who can access each infrastructure item and what data it contains, and which API functions access them or are hosted by them
- Continuously run **API Discovery** to identify changes in APIs and surface shadow APIs and rogue APIs
- Do the same for all services that are **integrated** with your API
- Thoroughly document **every aspect of your API**, including all authorization policies, error reporting, and security measures. Share this documentation with the team members responsible for testing and reviewing the security of the application
- To prevent unintentional data disclosure, avoid using production data for testing unless doing so **is absolutely necessary**

wallarm

API abuse, lack of resources and rate limiting

APIs that do not impose rate or resource limits are vulnerable to brute force or other DoS style attacks. For example, brute force attacks are common against authentication endpoints and make it easy for attackers to perform password stuffing or user enumeration attacks. In addition, attackers can use DoS techniques to overload API infrastructure.



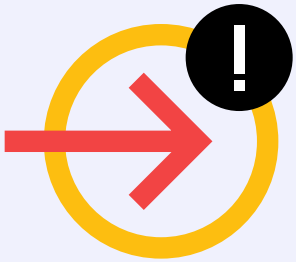
Limiting access to your resources

- **Implement rate limits** for every API endpoint that limit how many requests a client can make in a given period of time;
- **Implement** request size limits and limits to the size of submitted strings and arrays;
- Validate user submitted data **before** it is executed by your API functions;
- Use bot mitigation tools to **prevent abuse** by automated tooling;
- **Block traffic** from unwanted geographical regions, data centers, and Tor relay nodes;
- Only allow traffic to private API endpoints from **allow-listed** IP addresses;
- Keep known continuous attackers **in the block-list**.

wallarm

Injections

Unsanitized input from a malicious client can be used to execute arbitrary code on your infrastructure. SQL injections can result in attackers having full access to production databases, and shell injections have the potential to grant attackers control over application servers.



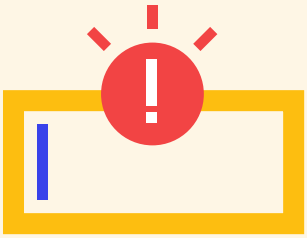
Best practices to avoid injection vulnerabilities

- [x] Validate and sanitize **all input** from the client;
- [x] Ensure all input is **properly escaped** using the correct syntax;
- [x] Use **API threat prevention tooling** that supports required protocols (REST, GraphQL, etc). Ensure that your solution provides poproper inspection for API calls and is a good fit for detecting injections;
- [x] Utilize your OpenAPI/Swagger schema **to validate incoming API requests** and block malicious requests (for example, by using an open source validator)

wallarm

Broken object level authorization (BOLA) / Insecure Direct Object Reference (IDOR)

Object Level Access Control issues arise if authorization checks are not performed for every function that could potentially be manipulated via user input by an untrusted party. For example, if an object is accessed by a unique ID specified by the end user, an attacker could change this ID in a malicious request to gain access to other objects the user should not have access to.



Steps you can take to fix broken authorization

- Implement an **authorization mechanism** that checks whether the logged in user has permission to perform an action;
- Use this authorization mechanism **in all functions** that accesses sensitive data;
- Use **randomly generated GUIDs** (as they are hard to guess) as object identifiers for user requests.

wallarm

Broken user authentication

Improperly implemented user authentication often renders other security measures obsolete as it does not provide a foundation of an authenticated user to build other security measures with. Technical flaws in a user authentication system can allow malicious parties to impersonate legitimate users. Some technical flaws could include using expired or leaked tokens/sessions, guessable or predictable authentication tokens, or otherwise broken credential verification before minting valid user sessions.

Protecting your API against attacks on your authentication system

- [x] Use commonly accepted standards **like OAuth and JWT** for the authentication process;
- [x] **Identify and document** all paths that can be used to authenticate with your API and ensure they are reviewed for possible credential leaks;
- [x] Do not return any **sensitive** information like passwords, keys, or tokens directly in API responses;
- [x] **Protect** all login, password recovery, and registration paths using rate limiting, brute force protection, and by adding lockout measures for abusive traffic sources;
- [x] Implement and use **multi-factor authentication** (MFA) wherever possible, and use revocable tokens where implementing MFA is not feasible.

wallarm

Excessive data exposure

For the sake of convenience, many developers expose all of the properties of objects through API endpoints. This is intended to allow front-end developers access to all of the required resources, but can result in unintended data exposure.

What your engineers can do to avoid unintended data exposure

- Define exactly which object properties are to be returned in your **API functions** rather than returning entire objects;
- Return **only the data the client requests** from your API functions rather than returning all available data and expecting the client to filter it;
- **Limit the number of records** that can be affected by a query in API functions to prevent mass updating or disclosure of database records;
- **Validate** API responses from a central schema that filters out object properties that should not be visible to the requesting user.

wallarm

Broken function level authorization

Overly complex and decentralized authorization policies make it confusing for engineers to implement the correct authorization for a given object or endpoint, leading to mistakes in authorization that can be exploited to access protected resources.

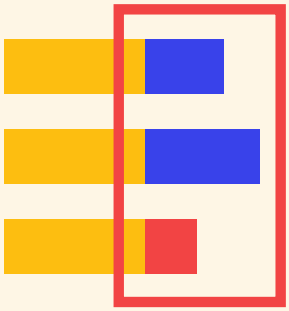
Important steps to fix authorization issues in your code

- [x] Ensure your authorization frameworks grant access **explicitly** to individual resources.
- [x] Ensure the default permission for all users for all resources is to **deny access**.
- [x] **Centralize** your authorization code so that it can be regularly reviewed and vetted, knowing that the review covers authorization wherever it is used in your API.

wallarm

Mass assignment

Mass assignment flaws allow attackers to modify objects by guessing property names or endpoint addresses that shouldn't be accessible, or by providing additional object properties through object relationships.



Prevent mass assignment attacks by implementing measures to validate input

- **Do not directly assign** user input to objects in your API functions or create or update objects by directly assigning user input;
- **Explicitly define** the object properties that the user is able to update in your API code;
- Enforce validation and data schemas so that only **approved** object properties will be used by your API functions.

wallarm

Security misconfiguration

Resource- and time-constrained engineers may use insecure default configurations for security software or appliances. Temporary configuration options used during development are commonly overlooked and make it to production, while attackers can take advantage of permissive cloud storage access policies, CORS policies, and overly-verbose error messages that provide access to, or information about, your API to exploit it.

Avoid exposing your API to attack by properly securing it

- Ensure your deployment process is **security hardened and well-documented** so that a secure hosting environment can be reproduced;
- Review your deployment configurations and process regularly, including any software dependencies used in your API, deployment and configuration files, and the security of your **cloud infrastructure**;
- **Limit all client interactions** with your API and any other resources (such as linked media) to secure, authorized channels;
- Only allow API access using **necessary** HTTP verbs to reduce attack surfaces;
- Set CORS policies for APIs that are **publicly accessible** from browser-based clients;

wallarm

Insufficient logging & monitoring

Poor application monitoring and logging allows attackers to get access to your data and infrastructure before they've been noticed, or without being noticed at all. Rigorous monitoring and accurate and informative logging are required to identify breaches and potential future threats, as well as to catch ongoing attacks before they can progress.

Your security engineers need to know about problems before they can fix them

- **Log** all authentication and authorization failures;
- Log request details that can be used **to quickly identify** the source of an attack using API security tooling;
- Properly format logs so that they can be filtered and reported with a **log management platform**;
- Treat logs as **sensitive** data, as they contain information on both your users and API vulnerabilities;
- Implement continuous monitoring of your infrastructure and tailor your monitoring reports to include the information that is **most important** to your API security.