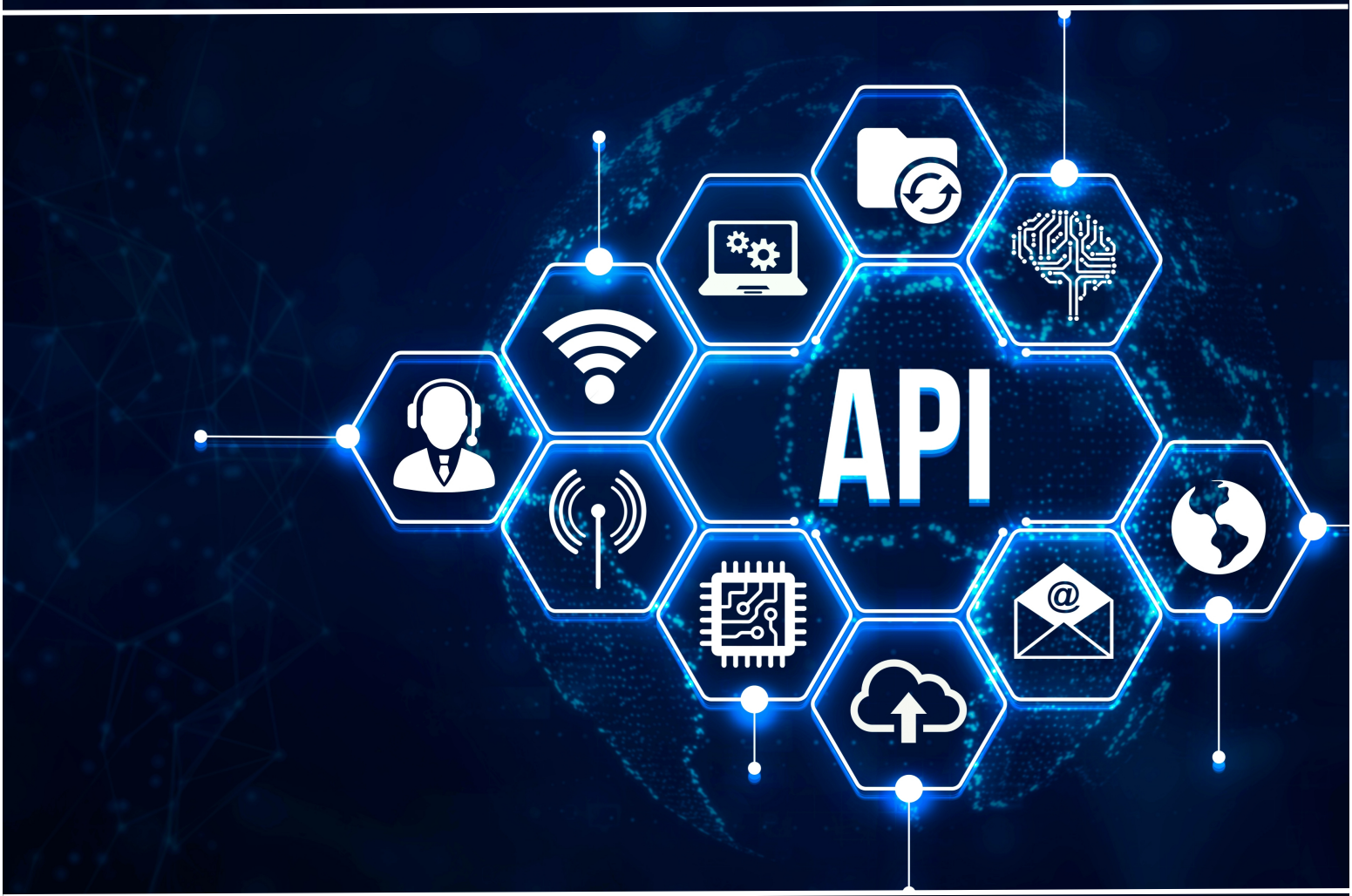
my redfort community

Bright

API Security: the Complete Guide

An eBook from Bright Security

March 2022



What is API Security

An Application Programming Interface (API) allows software applications to interact with each other. It is a fundamental part of modern software patterns, such as microservices architectures.

API security is the process of protecting APIs from attacks. Because APIs are very commonly used, and because they enable access to sensitive software functions and data, they are becoming a primary target for attackers.

API security is a key component of modern web application security. APIs may have vulnerabilities like broken authentication and authorization, lack of rate limiting, and code injection. Organizations must regularly test APIs to identify vulnerabilities, and address these vulnerabilities using security best practices.

This eBook presents several methods and tools for API security testing, and a range of best practices that can help you secure your APIs. You will learn:

OWASP API Top 10 Security Threats:

- Broken Object-Level Authorization
- Broken User Authentication
- Excessive Data Exposure
- Lack of Resources and Rate Limiting
- Broken Function-Level Authorization
- Mass Assignment
- Security Misconfiguration
- Injection
- Improper Asset Management
- Insufficient Logging and Monitoring

REST API Security vs SOAP Security

GraphQL Security

Methods Of API Security Testing:

- Test for Parameter Tampering
- Test for Command Injection
- Test for API Input Fuzzing
- Test for Unhandled HTTP Methods

Top API Security Open Source Tools:

- Postman
- Swagger
- JMeter
- SoapUI
- Karate
- Fiddler

API Security Best Practices:

- Identify Vulnerabilities
- Leverage OAuth
- Encrypt Data
- Use Rate Limiting and Throttling
- Use a Service Mesh
- Adopt a Zero-trust Philosophy

API Security with Bright

OWASP API Top 10 Security Threats

The increase of API-related security threats in recent years has prompted the Open Web Application Security Project (OWASP) to release the API Security Top 10, which helps raise awareness of the most serious API security issues affecting organizations These are:

API1:2019: Broken Object-Level Authorization

APIs often expose endpoints handling object identifiers. Any function that accepts a user input and uses it to access a data source can create a Level Access Control issue, widening the attack surface. You should carry out object-level authorization checks for all such functions.

API2:2019: Broken User Authentication

Attackers often take advantage of incorrectly applied authentication mechanisms. They may compromise an authentication token or exploit flaws in implementation to pose as another user, on a one-time basis or permanently. If the system's ability to identify the client/user is compromised, so is the overall API's security.

API3:2019: Excessive Data Exposure

Developers often rely on the client side to filter the data before displaying it to the user. This can create serious security issues—data must always be filtered at the server side, and only the relevant information should be delivered to the client side.

API4:2019: Lack of Resources and Rate Limiting

APIs often don't restrict the number or size of resources that the client/user can request. This can impact the performance of the API server, resulting in Denial of Service (DoS), and exposing authentication vulnerabilities, enabling brute force attacks.

API5:2019: Broken Function-Level Authorization

Authorization flaws often result from overly complex access control policies, or if there is no clear separation between regular and administrative functions. Attackers can exploit these vulnerabilities to gain access to a user's resources or perform administrative functions.

API6:2019: Mass Assignment

Mass assignment typically results from the binding of client-provided data (i.e. JSON) to a data model based on an allowlist, without proper filtering of properties. Attackers can modify object properties in a number of ways—they can explore API endpoints, read the documentation, guess object properties, or provide additional properties through request payloads.

API7:2019: Security Misconfiguration
Security misconfiguration often results from inadequate default configurations, ad-hoc or incomplete configurations, misconfigured HTTP headers or inappropriate HTTP methods, insufficiently restrictive Cross-Origin Resource Sharing (CORS), open cloud storage, or error messages that contain sensitive information.

API8:2019: Injection
Injection flaws (including SQL injection, NoSQL injection, and command injection) involve data that is sent to an interpreter from an untrusted source via a command or query. Attackers can send malicious data to trick the interpreter into executing dangerous commands, or allow the attacker to access data without the necessary authorization.

API9:2019: Improper Asset Management
Compared to traditional web applications, APIs typically expose more endpoints and thus require structured, up-to-date documentation. Issues such as exposed debug endpoints and deprecated API versions can increase the attack surface. This can be mitigated by creating an inventory of deployed API versions and properly configured hosts.

API10:2019: Insufficient Logging and Monitoring
Attackers can take advantage of insufficient logging and monitoring, as well as ineffective or lacking incident response integration, to persist in a system, deepen their hold and extract or destroy more data. It typically takes over 200 days to detect a persistent threat, and breaches are usually discovered by an external party—highlighting the critical importance of effective API monitoring.

REST API Security vs SOAP Security
There are two main architectural styles used in modern APIs:

> SOAP—a highly structured message protocol that supports multiple low-level protocols.
>
> REST—a simpler approach to APIs using HTTP/S as the transport protocol, and typically using JSON format for data transfer.
> Both types of APIs support HTTP requests and responses and Secure Sockets Layer (SSL), but the similarity ends there.

SOAP API Security:
- SOAP offers extensions to the protocol that address security matters

- SOAP is based on W3C and OASIS recommendations, including SAML tokens, XML encryption, and XML signatures.

- SOAP supports the Web Services (WS) specifications, which lets you use security extensions like WS-Security, which provides enterprise-grade security for web services

- SOAP supports WS-ReliableMessaging which provides built-in error handling

REST API security:

- REST APIs do not have any built-in security capabilities—security depends on the design of the API itself.

- Security must be built in for data transmission, deployment, and interaction with clients

- REST APIs do not have built-in error handling and need to resend data when an error occurs.

- A common architectural choice is to deploy REST APIs behind an API gateway. Clients connect to the gateway, which acts as a proxy, not directly to the REST API. This allows many security concerns to be addressed by the API gateway.

In conclusion, SOAP APIs are more secure by design, but REST APIs can be made secure, depending on their implementation and the architecture selected.

GraphQL Security

GraphQL is a query language that describes how clients can request information via an application programming interface (API). Developers can use GraphQL syntax to request specific data and receive it from a single source or multiple sources. Once a client defines the required data structure for a request, a server returns data using that exact structure.

Since clients can craft highly complex queries, the server must be ready to properly handle them. The server should be able to handle abusive queries from malicious clients, as well as large queries by legitimate clients. If the server does not handle these scenarios properly, the client might take the server down.

Here are a several strategies that can help you mitigate GraphQL security risks:

- Timeout – a timeout can help you defend against large queries. It is the simplest strategy because it does not require the server to know any details about incoming queries.

The server only needs to know the maximum time allowed per query.

- Maximum query depth – can help you prevent clients from abusing query depth. Maximum query depth is the analysis of a query document's abstract syntax tree (AST) to determine what is acceptable. The GraphQL server can then use this depth to accept or reject requests.

- Query complexity – query depth is not always enough to understand the scope of a GraphQL query. This usually happens when certain schema fields are more complex to compute than others. Query complexity can help you define the level of complexity of these fields, and restrict queries that exceed a complexity threshold.

- Throttling – the above options can stop large queries, but they cannot stop clients that make many medium-sized queries. For GraphQL, even a few queries could be too much to handle, if queries are expensive. You can determine the server time needed to complete each type of query, and use this estimation to throttle queries.

Methods Of API Security Testing

You can use the following methods to manually test your APIs for security vulnerabilities:

Test for Parameter Tampering

In most cases, parameters sent through API requests can be easily tampered with. For example, by manipulating parameters, attackers can change the amount of a purchase and receive products for free, or trick an API into providing sensitive data that is not authorized for the user's account.

Parameter tampering is often performed using hidden form fields. You can test for the presence of hidden fields using the browser element inspector. If you find a hidden field, experiment with different values and see how your API reacts.

Test for Command Injection

To test if your API is vulnerable to command injection attacks, try injecting operating system commands in API inputs. Use operating system commands appropriate to the operating system running your API server. It is recommended to use a harmless operating system command which you can observe on the server—for example, a reboot command.

For example, if your API displays content via a URL, you can append an operating system command to the end of the URL to see if the command is executed on the server:
https://vulnerablesite.com/view?name=userfile.txt;restart

Test for API Input Fuzzing

Fuzzing means providing random data to the API until you discover a functional or security problem. You should look for indications that the API returned an error, processed inputs incorrectly, or crashed.
For example, if your API accepts numerical inputs, you can try very large numbers, negative numbers, or zero. If it accepts strings, you can try random SQL queries, system commands, or arbitrary non-text characters.

Test for Unhandled HTTP Methods

Web applications that communicate using APIs may use various HTTP methods. These HTTP methods are used to store, delete, or retrieve data. If the server doesn't support the HTTP method, you will usually get an error. However, this is not always the case. If the HTTP method is unsupported on the server side, this creates a security vulnerability.
It is easy to test if HTTP methods are supported on the server side, by making a HEAD request to an API endpoint that requires authentication. Try all the common HTTP methods—POST, GET, PUT, PATCH, DELETE, etc.
Learn more in our guide to API security testing.

Top Open Source API Testing Tools

Securing production APIs, especially those that have a regular development and release process, requires automated tools.

The following open source tools can help you design security-related test cases, run them against API endpoints, and remediate issues you discover. They can also discover business logic vulnerabilities, which can also be an opening for attackers:

Postman is an API development platform. Its key features include:

- Automating manual API tests
- Integrating tests into the CI/CD pipeline
- Simulating expected behavior of API endpoints and responses
- Checking API performance and response times
- Enables collaboration between developers with built-in version control

Swagger is an open source toolkit that can help you create RESTful APIs. Its enables two API development styles:

- Top-down API design, letting you build an API in Swagger and then generate code from specifications
- Bottom-up API design, in which Swagger takes existing code and generates documentation about API operations, parameters and output.

JMeter is a load testing tool, which can also be used for security testing. Key features include:

- Inputting CSV files and using them for load testing—this lets you perform tests with different values to simulate risky scenarios and cyber attacks.
- Embedding API tests into the build process with Jenkins
- Advanced performance testing, with the ability to replay test results

Soap UI is a popular API functional testing tool. Its key features include:

- A large library of functional testing elements that let you automate API tests

- Fully customizable, provides source code so you can build your own features
- Easy drag and drop interface to create tests
- Lets you reuse existing load test or security scans for functional tests

In the pro package, lets you perform data-driven testing, simulating how users work with the API using spreadsheets or databases.

Karate DSL is a Java API testing tool using the behavior-driven development (BDD) approach. Its key features include:

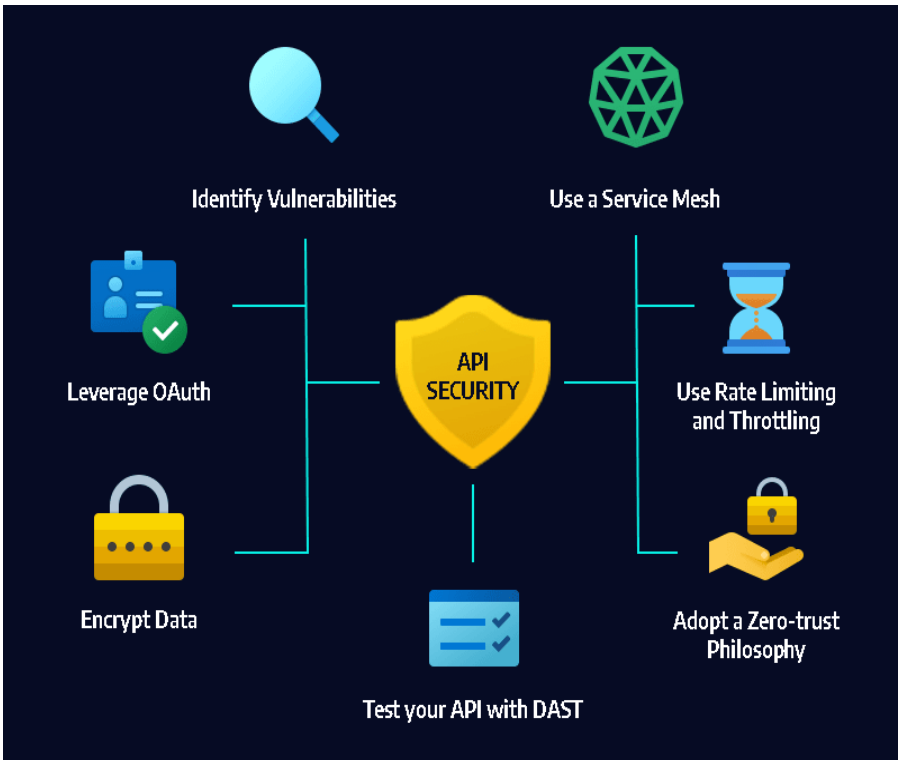
- Writing BDD for APIs with ready-made step definitions
- Generates standard Java reports
- Does not require Java knowledge to write tests for Java-based APIs
- Enables multi-threaded execution
- Supports switching configuration between staging and production

Fiddler is a tool that monitors and replays HTTP requests, with an API testing extension for .NET, Java, Ruby, and other popular frameworks. Its key features include:

- Debugging requests from any type of client—including Windows, MacOs, Linux, and mobile devices
- Tests cookies, cache, and headers in client-server communication
- Provides a convenient UI for grouping and organizing API requests
- Creates mock requests and responses with no code changes

API Security Best Practices

Use the following best practices to improve security for your APIs:



Identify Vulnerabilities

The only way to effectively secure an API is to understand which parts of the API lifecycle are insecure. This can be complex, especially if your organization operates a large number of APIs. It is important to consider the entire API lifecycle—the API must be treated as a software artifact, which goes through all the stages of a software product, from planning through development, testing, staging, and production.

Leverage OAuth

One of the most important aspects of API security is access control for authentication and authorization. A powerful tool for controlling API access is OAuth—a token-based authentication framework that allows third-party services to access information without exposing user credentials.

Encrypt Data

All data managed by an API, especially personally identifiable information (PII) or other sensitive data protected by compliance standards and regulations must be encrypted. Implement encryption at rest, to ensure attackers who compromise your API server cannot make use of it, and encryption in transit using Transport Layer Security (TLS).

Require signatures to ensure that only authorized users can decrypt and modify data provided by your API.

Use Rate Limiting and Throttling

As APIs become popular, they become more valuable to attackers. APIs are now a prime target for denial of service (DoS) attacks. Set rate limits on the method and frequency of API calls to prevent DoS attacks, and protect against peak traffic, which can affect performance and security. Rate limiting can also balance access and availability by regulating user connections.

Use a Service Mesh

Like API gateways, service mesh technology applies different layers of management and control when routing requests from one service to the next. A service mesh optimizes the way these moving parts work together, including correct authentication, access control and other security measures.

As the use of microservices increases, service meshes are especially important. API management is shifting to the service communication layer, and service meshes can provide automation and security for large deployments with multiple APIs.

Adopt a Zero-trust Philosophy

Traditionally, networks had a perimeter and elements "inside" it were trusted, while elements "outside" were not. Networks are no longer that simple, with insider threats becoming prevalent, and legitimate users often connecting from outside the network perimeter. This is especially true for public APIs with users from all over the world, accessing internal software components and sensitive data.

A zero trust philosophy shifts the security focus from location to specific users, assets, and resources. It can help you ensure that APIs always authenticate users and applications (whether inside or outside the perimeter), provides the least privileges they actually need to perform their roles, and closely monitors for anomalous behaviour.

Test Your APIs with Dynamic Application Security Testing (DAST)

Bright has been built from the ground up with a dev first approach to test your web applications, with a specific focus on API security testing.
With support for a wide range of API architectures, test your legacy and modern applications, including REST API, SOAP and GraphQL.

To compliment DevOps and CI/CD, Bright empowers developers to detect and fix vulnerabilities on every build, reducing the reliance on manual testing by leveraging multiple discovery methods:

- HAR files
- OpenAPI (Swagger) files
- Postman Collections

Start detecting the technical OWASP API Top 10 and more, seamlessly integrated across your pipelines via:

- Bright Rest API
- Convenient CLI for developers
- Common DevOps tools like CircleCI, Jenkins, JIRA, GitHub, Azure DevOps, and more

Bright Security: Our story

Gadi Bashvitz, COO and President of Bright Security tells us how it all started and why he thinks it's a game-changer

"Traditional Application Security Testing isn't keeping up and focuses on detecting known vulnerabilities. Legacy tools rely on a heuristics-based approach and lengthy and costly manual testing for finding new issues. This doesn't scale and results in substantial delays to remediation, putting your business at risk."

Bar Hofesh and Art Linkov decided to do something about it. They combined their experience in cyber security and biologically-inspired machine learning, creating Bright Security's AIAST technology, which automates a human's critical thinking process when detecting vulnerabilities.

"We think the results speak for themselves with a Dynamic Dynamic Application Security Testing (DAST) solution that fully automates AppSec testing at scale, allowing organisations of all sizes to stay ahead of even the most ruthless of hackers. It lets them comprehensively test, assess and improve their cybersecurity posture regardless of industry, including software, blockchain, FinTech, IoT, automotive, healthcare, and more."