Contents

[Security Best Practices 1](#_Toc158900692)

[1. Implement authentication and authorization. 1](#_Toc158900693)

[2. Use SSL/TLS encryption. 1](#_Toc158900694)

[3. Implement rate limiting. 1](#_Toc158900695)

[4. Use auditing and logging. 1](#_Toc158900696)

[5. Restrict access to sensitive data. 1](#_Toc158900697)

[6. Monitor and alert on anomalous activity. 1](#_Toc158900698)

[7. Use API gateways. 2](#_Toc158900699)

[8. Secure storage and encryption of data at rest. 2](#_Toc158900700)

[9. Use a Web Application Firewall (WAF). 2](#_Toc158900701)

[API Security Testing 3](#_Toc158900702)

[User Authentication Test 3](#_Toc158900703)

[Parameter Tampering Test 3](#_Toc158900704)

[Injection Test 3](#_Toc158900705)

[Unhandled HTTP Methods Test 3](#_Toc158900706)

[Fuzz Test 3](#_Toc158900707)

# Security Best Practices

1. Implement authentication and authorization.

**Pro tip:**Enable two-factor authentication (2FA) to provide an extra layer of security to ensure that only authorized people can access the system.

2. Use SSL/TLS encryption.

All communications between APIs and clients should be secured through an SSL connection or TLS encryption protocol like HTTPS. This ensures that all data sent over the wire is encrypted and kept safe from malicious third parties.

**What we like:**SSL/TLS encryption provides an extra layer of security to protect data in transit.

**Best for:**Any application that sends sensitive data over the wire or stores private user information.

**Pro tip:** Ensure your web applications use the latest and most secure version of HTTPS with TLS 1.3 protocol enabled to maximize security.

3. Implement rate limiting.

Set up rate limits tailored to your system's needs and adjust them depending on usage patterns.

4. Use auditing and logging.

It's essential to keep track of what users are accessing and what they’re doing with that information. Make sure to log every API request, and keep audit logs of user activity to ensure data security and compliance.

**What we like:**Auditing and logging help to keep track of user activity and prevent data breaches or non-compliance issues.

**Best for:**Any system that stores or transmits confidential information or personal data.

**Pro tip:**Make sure you have a comprehensive auditing and logging policy in place — and ensure it’s reviewed and updated regularly to keep up with the latest security threats.

### 5. Restrict access to sensitive data.

### 6. Monitor and alert on anomalous activity.

**What we like:**Setting up a system to monitor user behavior and alert you of suspicious activities.

**Pro tip:**Use automated tools to scan your systems regularly and set up alerts for any unusual activity. Enhance your system security by leveraging [popular vulnerability scanners](https://www.getastra.com/blog/security-audit/best-vulnerability-scanners/)through automated tools.

### 7. Use API gateways.

API gateways act as “gatekeepers” between the client and the server, protecting against malicious attacks before they reach their destination.

**Best for:** Any application that receives a large number of requests from outside sources.

**Pro tip:**Choose an API gateway with advanced filtering capabilities to maximize protection against potential threats.

### 8. Secure storage and encryption of data at rest.

Any data that’s stored locally should be encrypted to prevent unauthorized access. This includes any backups or snapshots of your data that may be taken.

**What we like:**Ensuring that all data stored locally is encrypted prevents access from unauthorized individuals or programs.

**Best for:**Any application where sensitive data is stored in local databases or backups.

**Pro tip:** Make sure you use multiple layers of encryption when storing confidential information to maximize data security.

### 9. Use a Web Application Firewall (WAF).

A WAF is a piece of security software that sits between your API and the internet, filtering out any malicious traffic before it reaches your server. It’s a great way to protect against DDoS attacks and other malicious activities.

**Best for:** Any web application or API facing external traffic, such as those exposed via public websites or mobile apps.

**Pro tip:**Select a WAF that can be easily customized to your specific needs — allowing you to set different levels of security based on different access points or user roles within the system. Then, make sure your WAF is regularly updated to ensure the latest security patches are applied—keeping your data and applications secure.

Ensure all the above best practices are implemented throughout your entire application stack, not just the API layer. This will ensure comprehensive security for all your data, regardless of where it resides.

# **API Security Testing**

## **User Authentication Test**

To test your authentication mechanisms, try sending API requests without proper authentication. See if your API responds with the correct error and messaging. Try this test with both no credentials and incorrect ones.

## **Parameter Tampering Test**

To run a parameter tampering test, try various combinations of invalid query parameters in your API requests and see if it responds with the correct error codes. If not, your API likely has some backend validation errors that must be resolved.

## **Injection Test**

Try injecting SQL, NoSQL, LDAP, OS, or other commands in API inputs. Then, see if your API executes them. These commands should be harmless, like reboot commands or cat commands.

## **Unhandled HTTP Methods Test**

Most APIs have various HTTP methods to retrieve, store, or delete data. Sometimes web servers will give access to unsupported HTTP methods by default, which makes your API vulnerable.

To test for this vulnerability, try all the standard HTTP methods (POST, GET, PUT, PATCH, and DELETE) and a few uncommon ones.

Send an API request with the HEAD verb instead of GET, for example, or a request with an arbitrary method like FOO. You should get an error code. If you get a 200 OK response, your API has a vulnerability.

## **Fuzz Test**

Fuzz testing should be one of the last steps of your API security auditing process. This type of testing pushes your API to its limits to discover any security issues that have yet to be revealed.

To achieve this, send a large number of randomized requests, including SQL queries, system commands, arbitrary numbers, and other non-text characters. Then, see if your API responds with errors, processes any of these inputs incorrectly, or crashes.

This type of testing will mimic Overflow and DDoS attacks.

An API manager or gateway tool will handle or help address the API security guidelines described above (including testing). Let's take a closer look at these tools below.

# **REST API Security vs. SOAP API Security**

Software developers may follow different architectures to build an API. The most popular are Representational State Transfer (REST) and Simple Object Access Protocol (SOAP).

REST APIs transfer data via the Hypertext Transfer Protocol (HTTP). Meanwhile, SOAP encodes data in [XML](https://blog.hubspot.com/website/what-is-xml-file?hubs_content=blog.hubspot.com/website/api-security&hubs_content-cta=XML) — a common markup language for storing and transferring information — and sends it via HTTP.

SOAP is more strict in its requirements than RESTful design, making this API type more challenging to build. However, it also tends to be more secure and better at preserving data integrity than other API designs.

Let’s break down their differences below.

## **RESTful API Security**

The RESTful protocol supports SSL to protect data when transferred but lacks built-in security capabilities, including error handling. It also does not support the Web Services (WS) specifications, so you can’t use security extensions like Web Services Security for enterprise-grade security.

That means the security of REST APIs depends on the design of the API itself or an API gateway.

## **SOAP API security**

Like RESTful, the SOAP protocol also supports SSL to protect data when transferred, but it goes further.

Not only does it include SAML tokens, XML encryption, and XML signatures (based on W3C and OASIS recommendations), which help secure the data being sent and received by SOAP APIs — it also supports the Web Services (WS) specifications.

This lets you use security extensions like Web Services Security for enterprise-grade security and WS-ReliableMessaging, which provides built-in error handling.