API Security Checklist

A guide to protect your APIs

CURITY

Some key things to keep in mind when designing and building APIs

Nowadays, we are witnessing the rise of the API economy where APIs play an essential role in business success. However, this development raises its new challenges. In 2017, Gartner predicted that by 2022, API abuses will be the most-frequent attack vector resulting in data breaches for enterprise web applications. 2020 – with its digital pandemic, "as insidious as Covid-19" – has provided more than enough additional arguments to support that prognosis. Therefore, it is as topical as ever to ensure the security of web APIs.

With this checklist covering API best practices, you'll be able to significantly reduce the threat of cyberattacks and secure your business.

API gateway in place: put your API behind a safe gateway

API gateways centralize traffic features and apply them to every request that hits your API. These features may be security-related, like rate limiting, blocking malicious clients, and proper logging. They may be more practical and business-related, like path- and headers rewriting and gathering business metrics. The failure to control these requests could easily result in a severe security threat.

Moreover, without a gateway, API providers would have to reinforce each endpoint with these features one-by-one. An API gateway eases the process of adding or fixing these features.

Finally, plenty of API gateway products are available on the market, which can be easily followed and implemented.

Central OAuth server to issue tokens

Access or refresh tokens should come from the server, not from APIs or gateways

The complexity of the processes involved explains the importance of choosing a central OAuth server as the exclusive method of issuing tokens. This procedure requires authentication of the client and user, authorizing the client, signing the tokens, and other operations. All these functions require access to different data, such as client information or the preferred authentication mechanism.
Furthermore, if many entities issue and sign tokens, it becomes increasingly challenging to manage all the credentials used to sign the issued keys.
Therefore, there is only one entity that can safely handle these processes — an OAuth Server.

JSON Web Tokens

Combine them with opaque tokens for external use

When APIs are concerned, using JSON Web Tokens (JWT) as access and refresh tokens is good practice.
Services that receive JWTs can leverage claim information to make informed business decisions. For instance, it might be used to determine whether the caller can access a particular resource and what data they can retrieve.

However, when tokens are exposed to third-party clients outside of your infrastructure, you should use opaque tokens instead of JWTs. The reason for this is that information in a JWT is available to everyone and easy to decode. This can raise privacy issues and challenges in ensuring that no sensitive data ends up in the JWT's claims.

Moreover, sharing JWTs with your third-party clients increases the chances that they will start depending on the data in the JWT. As a result, any consequent modification of claims in a JWT could result in a breaking change, requiring costly implementation upgrades in all third-party clients.

If you still want to use opaque tokens externally and benefit from JWTs in your internal communication, the two approaches might help you out. Both the Phantom Token Approach and the Split Token Approach involve an API Gateway translating an opaque token into a JWT. Additionally, you can read more about JWT security best practices and make sure you implement them as well.

Scopes for coarse-grained access control

Tokens with limited capabilities will secure you better

If stolen client credentials have limited scopes, an attacker will have much less power. Therefore, you should always issue tokens with limited capabilities. To do this, you should implement OAuth scopes.

Verification of token scopes can be done at the API Gateway to limit malicious traffic reaching your API. You should use Scopes to implement coarse-grained access control. This control could include checking whether a request with a given access token can query a given resource or verifying the client can use a given Content-Type.

Claims for fine-grained access control

Claims will defend your API

Broken access control is one of the top 10 API security vulnerabilities according to OWASP, so it's worth remembering that it can be prevented by the fine-grained access control at the API level.

The API should verify whether or not the request can reach the given endpoint. It should also check whether the caller has rights to the data and what information can be returned based on the caller's identity (both for the client and user).

This practice safeguards against situations in which attackers bypass the Gateway. If a malicious request slips through the Gateway, the API will reject it if the access control is architected right.

Zero Trust: it is not just a buzzword

Make sure your API limits trust to incoming traffic as much as possible.

The first step toward building zero-trust is using HTTPS for all API traffic. If possible, use HTTPS internally so that traffic between services cannot be sniffed.

Second, your services should always verify incoming JWTs, even if they are transformed from an opaque token by the Gateway. This again helps to mitigate situations where a request manages to bypass your Gateway, preventing a malicious actor from operating inside your company or infrastructure.

Libraries with JWT Validation

They will help to limit the risk of mistakes and bugs

Proper JWT validation is crucial for the security of your APIs. If every team implements their own JWT validation solution, you risk increasing overall system vulnerability. Mistakes are more common, and it's difficult to fix bugs.
Instead, create a company-wide solution for JWT validation, preferably based on libraries available on the market and tailored to your API's needs.
Standardizing a company-wide JWT validation process will help guarantee the same level of security across all your endpoints. When issues arise, teams can resolve them more quickly.
For security-sensitive tasks like JWT validation, quick threat resolution is incredibly important.

JSON Web Key Sets for Key Distribution

Opt for a key rotation

To verify a JWT's integrity, an API must access a public key. You can accomplish this in a couple of ways: you can hardcode the key's value or query some endpoint at your service startup and cache the result.

The recommended method is to obtain a key from a JWKS endpoint exposed by the OAuth server. The API should cache the downloaded key to limit unnecessary traffic and query the JWKS endpoint again whenever it finds a signing key it doesn't know.

This allows for a simple key rotation, which the OAuth Server can handle on-demand without impeding the API services. Using key sets instead of keys also allows a seamless key rotation for the clients. The OAuth Server can begin issuing new tokens signed with a new key and validate existing tokens as long as the old public key is part of the key set.

Different authentication methods

Do not mix authentication methods for the same resources

Don't mix authentication methods at different security levels, such as Basic Authentication and Multi-Factor Authentication. If you have a resource secured with a higher level of trust, like a JWT with limited scopes, but allow access with a lower level of trust, this can lead to API abuse. In some cases, this could be a significant security risk.

Continuous check for potential abuse

Catch unwanted behavior before the actual break

Just because your API security isn't breached doesn't mean that everything is fine. You should gather metrics and log usage of your API to catch any unwanted behavior. Watch out for requests iterating over your IDs, requests with unexpected headers or data, customers creating many clients to circumvent rate limits, and other suspicious cues. Losing data due to API abuse can be just as harmful to your business as hackers break through security.

Overall Protection

Do not leave any of your APIs unprotected

Don't forget your internal APIs. They should also have protections implemented to ensure that the API is protected from any threat from inside your organization.

It's not uncommon that APIs are created for internal use only and made available to the public later on. In such scenarios, proper API security may be overlooked. When published externally, the API becomes vulnerable to attacks.

Also, security by obscurity is not recommended. Just because you create a complicated name for an endpoint or Content-Type does not mean the API will be secure. It's only a matter of time before someone finds the endpoint and abuses it.

Audit 4ever

Ensure you have additional eyes checking your APIs

Maintaining high standards for your APIs, both from a security and design point of view, is not a trivial task. Therefore, consider splitting responsibility between different groups of people and have other teams audit your APIs.

There are different approaches to setting up governance over your API. You could have a dedicated team of API experts review the design and security aspects or create a guild of API experts picked from different groups to offer guidance.

Conclusion

Maintaining high-standard API security is an important task.

As we've seen above, there are many technical strategies to consider when designing your authorization processes, which, if undermined, can directly affect API security.

A stronger foundation is only made possible with a secure, centralized OAuth server responsible for certificate generation and handling claims. Many suggestions also revolve around treating internal APIs with the same care as public-facing endpoints.

By following these best practice measures, you can safeguard your APIs and thwart unwanted behaviors.

To dive deeper into the issue, check out Curity's resources on API security and other identity and access management topics.

CURITY

Address
Curity AB
S:t Göransgatan 66
112 33 Stockholm
Sweden

Website
curity.io

Email
info@curity.io

Twitter
@curityio