A Modern CISO's Guide to API Security

Authors: Jonathan Care, Lionfish Tech Advisors

As a CISO, your role isn't to dig into the code and fix issues yourself, as much as you might want to. Your role is to set policy. The implementation of that policy comes down to making the right training available, ensure management support from the executive team, and instill a sense of personal security responsibility in each employee. Given than majority of traffic and revenue now goes through APIs, it also means ensuring that your organization has the right API security tools in place that can assist in assessing API risks, provide remediation and protect your APIs from attack.

API Security is a confusing market with several point approaches and solutions. In this Modern CISO Guide to API Security, we are going to describe the risk of APIs, API security best practices, and the requirements needed in an API Protection solution that can protect organizations large and small from persistent API threats. We hope you find this guide useful in your API Protection journey.

What's an API?

API stands for application programming interface. In less fancy terms, it's a gateway that allows two pieces of software to interact in some meaningful and standardized way.

For example, if you run a music service that can suggest hot new releases to users, you might want to allow third party services to poll your 'recently added' section with a list of user preferences and parameters. The gateway that handles that data exchange is an API.

Companies use APIs on a daily basis for all manner of things ranging from software updates, to news aggregation, to triggering certain automated responses via a **REST** (Representational State Transfer) API.

Are APIs Natively Insecure?

That depends on how they are implemented. With the right safeguards and underlying protocols, APIs can be secure and incredibly useful. With a lax implementation, or built on the wrong foundation or premise, an API is literally an open door to your data.

This is why all of the policies that you lay out will encourage API best practices that conform with core cybersecurity best practices.

What are the Main Risks With Poorly Protected APIs?

The most common API risk for organizations are shadow APIs. Shadow APIs are unmanaged APIs that are unknown to the security team. They are often quickly implemented by internal development teams in a cloud provider such as AWS without every informing the security team of their existence. Often there is no API protection that is enabled, allowing an attacker to easily exploit an application and exfiltrate sensitive data.

The second most common risk with unprotected APIs is **overexposure of sensitive data**. In other words, the API is giving too much access, too much information, or it's exposing hints that can be used to attack the API host or underlying architecture. In the same way that you wouldn't give a guest unfettered access to one of your databases, you can't allow APIs to give out everything that some anonymous user asks for.

The next most common API risk is **improper authentication or authorization**. This might manifest at the object level, giving the API itself too much authority. For example, in a broken object level authorization (BOLA) attack, a user is able to access the resources of another user on the system since authentication and authorization has not been properly implemented within the API application. You might be surprised but this is one of the most common attack vectors seen in API applications.

Another risk is that **the API is vulnerable in the face of DDOS attacks**. APIs without any type of protection can be pressured into processing false requests that squeeze out legitimate requests made by users, affecting the API application's overall performance. APIs require proper rate limiting and DDoS mitigation. The adoption of an DDoS mitigation provider can provide around the clock protection that ensures that your applications and business are never disrupted.

Mass assignment[1] can be a problem if the software development framework supports this feature. These software frameworks leave options on without requiring the developer to identify the specific data to be used in HTTP request parameters in program code variables or objects. Attackers can in turn use this methodology to create new parameters that create or overwrite new variables or objects in the application that was never intended. Leaving mass assignment options turned on without identifying the specific data to be processed can open the API up to new attack vectors.

Other risks include *improper logging and monitoring*, *insufficient automated testing*, *bad asset management*, and *overly verbose error messages*. While these apply to more than just APIs, they need to be kept in mind.

Best API Security Practices

These are some of the standards, development practices and policies that you must insist upon whenever an API is implemented:

- **Use well tested authentication and authorization standards.** Hundreds of thousands of hours have been poured into the likes of **OAuth 2 and OpenID Connect**. Don't reinvent the wheel when it comes to authentication and authorization standards. Use an open standard that is widely understood and trusted. Trusting your application development team to develop their own authentication and authorization schemes will result in an explosion of vulnerabilities ready to be exploited. There are countless stories of companies that created in-house authentication standards, only to have the one person who fully understood the system quit or retire suddenly. Don't become the next cautionary tale.
- **Encryption first.** Transport Layer Security (**TLS**) was created in order to improve the security of transmitted data. If you aren't using it to protect communications with your API, this is likely to be seen as a design pitfall and should be reviewed. It is a widely accepted best practice to use TLS to encrypt communications in transit and use proper encryption between the API and data stored in the backend. The use of unencrypted data stored in the backend should be carefully reviewed. A common attack vector is to capture data stored in temporary files, in particular sensitive transaction data such as authentication credentials, payment card data or other sensitive information.
- **Use a proven IAM solution.** As many cloud architectures are based on identity-first security principles, the use of a full-fledged identity access management (IAM) fabric is recommended. Many IAM providers will support your API security and authentication needs and support multiple authentication techniques. It's critical to ensure that all API endpoints within an API application are authorized by an IAM service, minimizing the risk of a rogue API with no authentication that can act as a doorway into your application for an attacker. Ensure that your development teams settle on one IAM provider so that all API applications are authenticated and authorized in the same way.
- **Scan and validate all input.** Injection attacks are a permanent threat. Even the simplest API can be leveraged in an adversarial attack. All input vectors need to be sanitized to ensure that they do not include malicious input. A well-recognized best practice is to use an existing and well tested library for the kind of validation needed. You should implement XML schema validation, JSON validation, and SQL validation wherever it is relevant. Attackers will naturally gravitate towards API application interfaces that lack appropriate input validation, which can be easily surfaced through the use of automated scanning tools.
- **Keep API error messages short.** Error messages should follow standard security policy, which means they should be short and factually correct, but not give any ammunition to a potential hacker. For example, a user providing the wrong password should get a generic login error, not an 'incorrect password' error: the latter allows them to fish for valid account names. Keeping your error messages succinct, ensures that extra information doesn't unnecessarily seep out that can give an attacker the extra edge to move deeper into your application.
- **Protect your API keys.** API keys should never be directly embedded in the code. That's a recipe for disaster. Similarly, they shouldn't be hard coded into the application's source repository. Instead, use environment variables or files that lie outside of the application's source code. Alternatively, use a secrets management service.
- **Create a central API inventory.** APIs shouldn't just pop up overnight, unannounced. Every API that your company runs or is even partially responsible for needs to be registered in an API database. An API attack surface management tool should be used to discover your entire API attack surface that includes both managed and unmanaged(shadow) APIs. This information should be populated in your API inventory database. Each API should be assessed for risk and vulnerability analysis to understand your organization's overall security posture. Additional information such hosting providers and the department and asset owners of the responsible APIs should be included. This API inventory should be continuously updated in real time with all changes made to ensure that all information is available and relevant.

1 From OWASP: "Software frameworks sometime allow developers to automatically bind HTTP request parameters into program code variables or objects to make using that framework easier on developers. This can sometimes cause harm. Attackers can sometimes use this methodology to create new parameters that the developer never intended which in turn creates or overwrites new variable or objects in program code that was not intended. This is called a Mass Assignment vulnerability.

- **Test early, test often.** As much as manual session-based testing can be useful and wonderful, *API security requires automated testing*. Every time the service comes online, every time the server is rebooted, every time a load balancer spins up another instance, and every time there's a patch or code change, the APIs need to be fully tested. Create your tests in JMeter, Postman, REST Assured, or the automation suite of your choice. Then add it to your Jenkins task list and put it on a periodic timer. If the automated test fails, you do not deploy. A failed API test should be considered a significant finding that is likely to expose the system to unauthorized access, unless further investigation proves otherwise.
- **Patch your software, firmware, and infrastructure.** This should go without saying, but in case it isn't obvious, the platform that your API runs on needs to be kept up to date at all times. Regularly check for updates to the libraries, suites, and dependencies that the server relies upon. The same goes for the underlying operating system and firmware, as well as the network tools and applications that keep you safe. Remember to run the aforementioned automated tests after any updates.
- **AI or ML monitoring techniques for APIs.** While AI or ML techniques may seem to be in the realm of advanced computer science, these are tools that are in common deployment in leading vendor solutions and mature end user security operations teams. API-specific artificial intelligence, such as **CQAI**, can be implemented that will pull in data from a broader dataset than just your application in order to make more intelligent decisions on malicious traffic. By leveraging normal operational data and attack vector data from hundreds of third-party sources, your API protection can react to even the most subtle threats immediately. Hundreds of eyes all around the world are better than relying on a single perspective.
- **Check OWASP for new or emerging threats to API security.** OWASP is devoting far more focus to API security threats and is continuously revising its threat ranking process to reflect that. The OWASP API Top 10 provides a structured way to understand the most critical API vulnerabilities that are facing organizations around the world. This ranking is periodically updated by security experts as they identify the most critical vulnerabilities within API applications that could be exploited by attackers.
- **Set sane rate limits.** To safeguard against DDOS attacks as well as systematic database replication, use a rate limiter. This limiter will pay attention to request frequency as well as setting CPU and memory thresholds within the API itself. If any of these limits are broken, the API's usage will be throttled accordingly. Use caching, sideloading, and load balancing as needed to stay up to date with your API's popularity. Just don't mistake a systematic scraping of your entire database as 'popularity'. Have the appropriate checks in place.
- **Add critical APIs to your disaster recovery and business continuity plans.** Associated with the central API registry, identify the APIs that you, and your customers or clients, cannot live without. Anything mission critical needs to be added to your DR/BC planning as priority one for regional mirroring, backup and recovery, and so on. Ensure there is a process in place to ensure that if an API host does go down, that the host will be reinitiated and start to services API requests.
- **Provide API security training and leverage online webinars.** Make sure that API security courses and webinars qualify as continuous training for your coders, testers, and dev-ops personnel. As long as a certain percentage of them are willing to become experts in API security, that knowledge will filter through at the product level. Check out **Cequence's webinar series** for some excellent examples.

API Security Solution Requirements to Protect Your APIs

API Discovery and Runtime Inventory

CISOs must understand the importance of implementing automated methods to identify the APIs their organizations utilize, monitor sensitive data movement, and detect changes within the ecosystem. Oftentimes, a company's API footprint may extend beyond what its developers are aware of, making visibility and discovery tools essential. IT security teams can only protect what they can see.

Unmanaged APIs, also known as shadow APIs, may be present within a company's application ecosystem, and they need to be detected alongside known APIs to ensure that OpenAPI specifications are enforced. API visibility tools should be vendor-neutral and capable of integrating with all existing infrastructure components, including API gateways, proxies, and controllers.

Threat Detection and Mitigation

API threat mitigation is a critical component that ensures that there is always protection enabled that can block API threats in real-time. Threat mitigation always starts with accurate threat detection. An API security tool should be able to understand the business logic of an application and the relevant security use cases. This ensures that any threat detection is not a one-size fits all approach that can lead to very high false positive rates, missing critical API attacks that actually matter.

What follows is threat mitigation, which is the ability to block API threats in real-time as they are discovered. This is a must have capability for organizations with mission-critical API applications to ensure they can block threats before they disrupt the business. A mature API security solution should have a flexible response options for malicious API traffic, that include blocking, logging, deceiving, or rate-limiting an attackers' access. Threat mitigation tools should be customizable to allow companies to defend against threats without obstructing legitimate traffic.

Design and Architecture

API security solutions should be designed based on an organization's specific needs. Considering the diverse range of applications for API-based development, this could mean deploying a cloud-based software-as-a-service or implementing an on-premises solution in a data center. Hybrid deployments, featuring on-premises data collection for security and compliance and a cloud-based control plane, also play a significant role.

API security tools should analyze only the necessary sensitive data to fulfill their purpose. Ensuring compliance with regulations such as the General Data Protection Regulation is vital to avoid privacy liabilities.

Network Integrations

An all-encompassing API security deployment should integrate with various network infrastructures to provide visibility into all types of API traffic, both inline and out of band. Integration with content delivery networks or load balancers allows organizations to analyze additional information.

Both internal and external APIs should be included in API security efforts, as either could expose an attack surface that compromises sensitive data. Integrations with gateways, proxies, load balancers, controllers, and more can help achieve this level of coverage.

Ecosystem Integration

Incorporating API development workflows into API security solutions helps minimize the risk of introducing new vulnerabilities into production. The API security checklist concludes with integrating DevOps workflows and existing security tools.

With these integrations, developers can make API security visibility and risk mitigation an integral part of their continuous integration and deployment workflows. The highly automated and efficient features promote more secure application development without hindering progress.

Conclusion

Our reliance on APIs is only going to grow in the near future. APIs are powering today's mission-critical applications across the largest and most complex businesses. As a result, attackers are recalibrating their attack campaigns to target API applications. CISOs, especially those working in finance and retail industries must remain vigilant in ensuring that their customer's sensitive data is protected at all times, especially when it is protected by regulation. Applications are now routinely cloud-native, with developer teams building complex architectures through the interlinking of APIs to generate increased value. This complexity can breed opportunities for an attacker to exploit applications and sensitive data.

As a result, it's important to ensure that the security for these APIs is airtight. This means that security standards need to be kept high and security best practices should never be compromised and always followed. This should be reinforced with an API protection tool that can discover your APIs, surface risks, and provide real-time protection against all persistent threats across organizations large and small, ensuring there is no disruption to your business or to your customers.