APIsec

Best Practices for API Security

Dan Barahona
dan@apisec.ai
apisec.ai

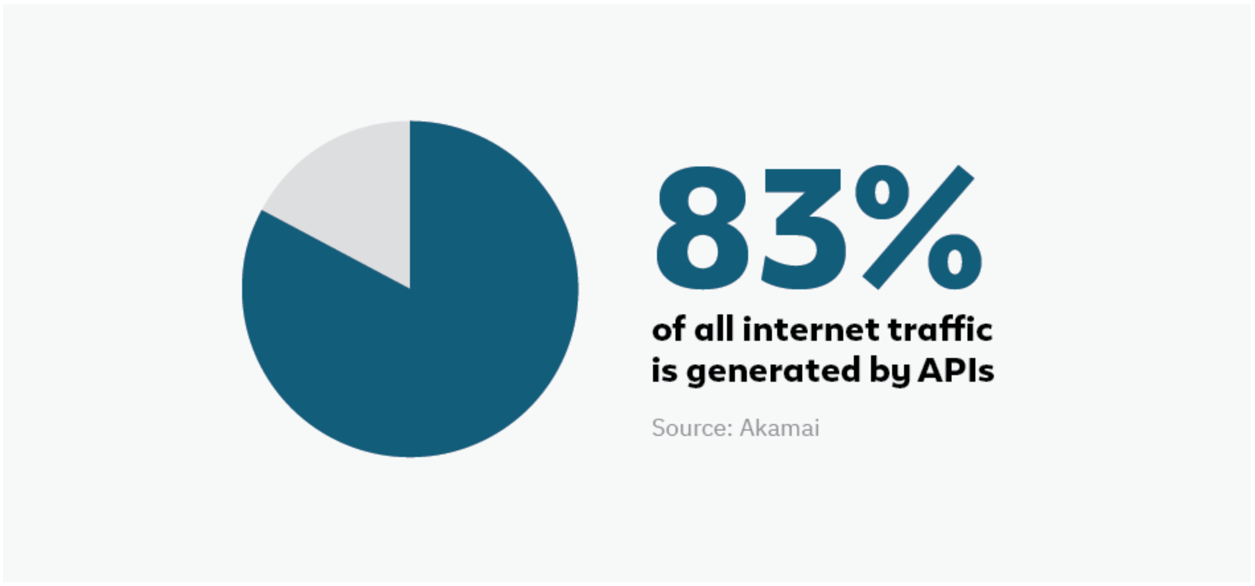
A decade ago, securing a company's web interfaces was a straightforward affair, and website vulnerabilities were well understood. Periodic scanning and manual penetration testing provided some assurance that a company's web and mobile applications were resistant to hacking. Since then, the world of web applications has changed radically, with Application Programming Interfaces, or APIs, transforming how organizations build, manage, and scale their web and mobile services.

Hackers have taken notice of the prevalence of APIs, and the existence of serious vulnerabilities within these interfaces. Daily, new API-based breaches occur with hackers taking advantage of undiscovered loopholes in API logic, allowing malicious access to sensitive data. This paper examines the challenges and best practices for securing APIs.

Adoption of APIs

Around 2010, most IT security professionals viewed the world as divided between their internal network and the dangerous world beyond. Providing an API for outsiders to make use of internal network services was uncommon. Two events occurred that changed the landscape. With soaring acceptance and popularity of Amazon Web Services and other competing cloud offerings, the mentality of maintaining strong boundaries between the inside and outside world fell from favor.

RESTful APIs became widely adopted, using HTTP verbs and endpoints that resembled the organization of web-based sites. Developers and IT shops found an easy familiarity with a tech that has massive potential. By 2019, Akamai, the leading content delivery network, reported that 83% of all internet traffic was now generated by APIs.



Emergence of APIs as a Security Risk

Predictably, as APIs gained traction as an ideal way to build applications and expose data and functionality with users and partners, attackers also discovered malicious opportunities in targeting APIs. In fact, APIs are considered by many to be among the most serious security threats organizations face, as APIs provide direct lenses into highly sensitive data and functionality.

90%
of web app attack surface area are APIs
Gartner

90%
of breaches targeted web applications
verizon

2022
APIs will become "most frequent attack vector"
Gartner

The problem is that web applications remain a primary target for breaches (90% according to Verizon) and that APIs now make up 90% of the web app attack surface area. This led Gartner to forecast that by 2022, APIs will become the "most frequent attack vector".

OWASP API Security Top 10

The Open Web Application Security Project (OWASP) is a U.S.-based non-profit dedicated to web security, application security, and vulnerability assessment. The OWASP Top Ten Web Application Security Risks is highly regarded as the go-to standard for testing and securing web applications. In response to the escalating breaches of APIs, the OWASP organization released a separate guidance in 2019 - the API Security Top 10.

Of OWASP's top ten API vulnerabilities only one involves a classic security attack vector, number eight, which covers various SQL injection vulnerabilities - acknowledging the unique challenges of securing API-based applications. Four of the top five vulnerabilities all involve logical errors in the software stack that are unique to each organization and each API, and are difficult to identify with traditional testing methodologies.

More importantly, the API security guidance recognizes a fundamental difference in how APIs are breached, typically via authorization and authentication, not classical web application security issues such as injection attacks, cross site scripting, buffer overflows, etc.

In fact, the common thread among most API breaches is that these are not classic security attacks and vulnerabilities. These breaches are the result of business logic flaws and loopholes in the API itself. Developers are taught to never trust user data, but APIs allow attackers to modify request properties in unexpected ways that are difficult to test for given the huge number of possible attack scenarios. In the rush to push more code, functionality and fixes into production, developers often introduce unintended access vulnerabilities as well. Here are some specific examples:

OWASP API Security #1: Broken Object Level Authorization

This represents the single most common, and often most serious, vulnerability for APIs. Broken Object Level Authorization vulnerabilities relate to what restrictions exist in the API logic to prevent User A from accessing User B's data - or any unauthorized data. In traditional web and mobile applications, the interface itself controls what data is presented and what the user is permitted to see. With APIs no such graphical interface exists and the API will return all the information that the function call allows.

UNITED STATES POSTAL SERVICE

Real-world example

The US Postal Service's "Informed Delivery" API allowed an authorized user to access, via API, all their shipments, delivery status, account information and more. While this API required users to be properly authenticated, it did not include controls over which customer's data could be accessed. With minimal changes in API requests, a user could access any other user's shipment details and account information. In this way data one user was able to harvest sensitive information on 60 million users.

OWASP API Security #2: Broken Authentication

API Authentication refers to the implementation of strong, effective, properly configured authentication to access data. Inadequate, or “broken,” authentication may be as basic as lack of any authentication at the API layer, or authentication that employs weak password, policies, lockout, etc. In other cases it can be a lack of token encryption, passing tokens in URL strings or allowing credential stuffing tactics to proceed.

Fizikal

Real-world example
A hacker discovered they could request password resets for the fitness platform, Fizikal, via API by supplying a phone number. Valid phone numbers would return a different response, so they simply iterated through all possible phone numbers to identify all valid accounts. The reset code was a simple 4-digit code, which could easily be brute-forced, giving the attacker access to any account.

OWASP API Security #3: Excessive Data Exposure

This vulnerability focuses on APIs that return more data, fields and information than the specific use requires. Many web and mobile apps rely on API calls that return more data than necessary, and then present only what's needed in the user interface. This approach relies on data controls in the UI, but exposes unfiltered results in direct API calls.

Real-world example

The electronic payments platform, Venmo, employed an API used by the corporate homepage to present a list of real-time transactions. A hacker discovered the API and was able to call the function directly, with no authentication required (a violation of #2 - Broken Authentication). However, not only did the API allow unauthenticated users, it also returned full transaction details. The hacker was able harvest over 200 million transactions including details of sender and recipient, goods and services description, and total amount.

OWASP API Security #4: Lack of Resources & Rate Limiting

In light of the machine-to-machine nature of APIs, these resources are especially vulnerable to highly automated, high volume denial of service attacks. These attacks can come from many different IPs, target different API functionality and data. Without restrictions on frequency and volume of API requests attackers can brute-force password resets, harvest user information, impact performance, and download massive quantities of proprietary information.

Instagram

Real-world example
Instagram allowed password resets by sending a 6 digit code to the account owner's device, giving the user 10 minutes to provide the correct code. Instagram also limited code submissions to no more than 200 attempts per IP address. However, there was no restriction on the number of attempts per user account. A hacker discovered the API for submitting security codes and setup a farm of servers on AWS, each with a different IP address, allowing the hacker to run through all 1 million possible 6 digit codes in less than 10 minutes and take over any account of their choosing.

OWASP API Security #5: Broken Function Level Assignment

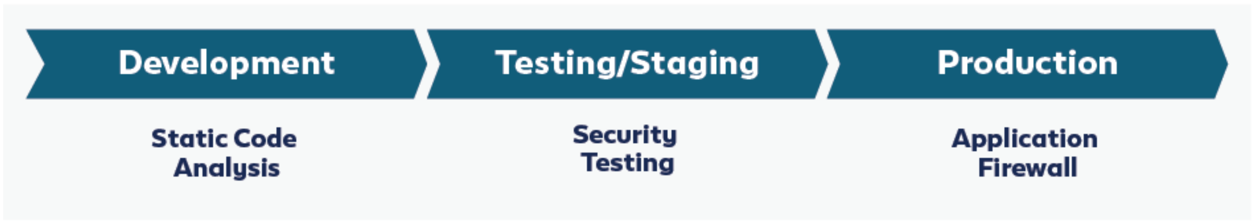
Broken Function Level Assignment refers to what functional capabilities and API allows users to access and execute. Every API function, or endpoint, generally supports a range of methods - including PUT, POST, GET, DELETE and others. Organizations must carefully consider which specific endpoints and methods need to be enabled for users and third parties. For example, an API might allow a user to GET their current billing statement. It should not allow the user to DELETE the statement or PUT the balance to zero.

Real-world example

The popular dating app, Bumble, enforced significant user access and functionality restrictions within their mobile and web UIs. However, those UIs use a suite of APIs to interact with backend systems - these APIs had far fewer controls and restrictions. One user discovered he could not only query APIs to pull information on every other user on the platform - but the API also permitted him to change his account settings and permissions. He could even turn on all premium features without paying, and use them without volume restrictions.

Approaches to API Security

The OWASP organization recognized the fundamental differences in security risk for APIs versus web and mobile apps. Similarly, the widely used tools and techniques used to secure web and mobile apps do not offer the same protection for APIs. New approaches are required for securing APIs - these approaches generally fall into three categories:



1. Static Code Analysis: building secure APIs during development
2. Security Testing: security and vulnerability testing
3. Application Firewall: protecting live APIs with inline/traffic solutions

Building security testing into a company's CI/CD pipeline is a common first line of defense. This approach focuses on the code itself, adding Static Analysis Security Testing (SAST) and other code quality solutions to the continuous integration workflow. There are well established vendors in this segment, and SAST can help enormously with ensuring the quality and maintainability of a codebase and to identify common, well-known coding issues and vulnerabilities. However, static code analysis is incapable of identifying the types of logic flaws that lead to major API breaches and data loss.

Operations often deploy a second line of defense by deploying Web Application Firewalls (WAF) and API-aware traffic inspectors to the production API environments. These firewalls analyze network traffic and employ heuristic techniques to watch for common attack patterns. API-aware firewalls can go a step further, looking for API-specific anomalies, such as preventing strings from being passed to an API that should only receive integers. This is a capability that would be useful across all APIs.

However, such technologies would not generally be able to identify an API user that is attempting to access data belonging to another user. This requires a level of user visibility and session awareness that API-aware firewalls lack, despite attribution techniques including IP addresses, packet headers, behavioral patterns, and user-agents that are employed in web application firewalls.

The primary issues with code analysis and API firewalling is that these technologies focus primarily on classic security-oriented vulnerabilities such as SQL injection attacks, cross-site scripting, buffer overflows, etc. As we saw earlier, these are not the causes of the vast majority of API breaches - these breaches are a result of the business logic flaws that are unique to each API and cannot be detected or prevented with standard approaches.

Lastly, companies invest in manual penetration testing by security professionals or internal Red Teams. Pen-testing relies on humans to understand API code, craft tests to identify vulnerabilities, execute the tests, and then interpret the results. This manual approach is, by definition, slow (weeks, not minutes), reactive (often a once or twice a year effort), and costly. Security vulnerabilities are only discovered after they have reached production. Despite these drawbacks, manual pen testing approaches security from the mindset of an adversarial attacker, adding depth in defending against security attacks.

Meanwhile, organizations continue to operate at the speed of DevOps, with new code pushed to production every day. As a result, defects make it into production, creating vulnerabilities and leading to serious breaches as described earlier.

Challenges of API Security Testing

With the move towards API-driven applications and functionality, the challenge of security testing code gets even harder. There are three main issues: coverage, scale, and speed.

Coverage

The ultimate goal of security testing is to make code less vulnerable to attack, misuse and exploits. To achieve this testing must cover every corner of the API's capabilities, not just what is expected. Corey Ball, author of the upcoming book "Hacking APIs" puts it succinctly, "You can design an API you think is ultra-secure, but if you don't test it, then a cybercriminal somewhere is going to do it for you." So the testing scheme needs to look at every API endpoint and method, and consider all the possible ways your API can be used, not just the likely ways.

Scale

The coverage problem leads to the scale problem. How do you create test scenarios to cover the entire range of API functionality. Consider a relatively lightweight API with, say, 50 endpoints. Each of those endpoints can support multiple POST, GET, PUT and DELETE methods. Quickly you're already up to ~250 endpoint-method permutations. And then consider the myriad API breach categories as described by the OWASP API Security Top 10 - pushing testing requirements to thousands of unique attacks. Testing needs to cover all these permutations and scenarios, otherwise vulnerabilities will make it to production waiting to get discovered by someone else.

Speed

Speed in CI/CD time means seconds or minutes. When fixes and new functionality need to move to production, testing cannot delay progress. However, given the complexity of APIs, the breadth of scenarios to cover, testing speed is more often measured in weeks or months.

A Better Way:
Automated API Security Testing

The coverage problem is multiplied by the scale problem - how to create test scenarios to cover the entire range of API functionality. Consider a relatively lightweight API with, say, 50 endpoints. Each of those endpoints can support multiple POST, GET, PUT and DELETE methods. Quickly you're already up to 200+ endpoint-method permutations. And then consider the myriad API breach categories

The APIsec approach to API security begins with learning the API: cataloging all available endpoints and identifying supported methods. This approach is 100% automated and allows critical API vulnerabilities to be addressed before a company's product reaches Production. This allows approach offers major advantages to static code analysis, API firewalling, and manual pen testing:

1. **Critical Vulnerability Detection** - discovers vulnerabilities missed by static analysis and firewalls, including business logic faults and access control issues.

2. **Automatic Test Creation** - automatically creates thousands of test sequences with no manual effort. This frees a team from the necessity of dreaming up every possible test scenario and hand crafting tests, potentially saving thousands of hours of effort.

3. **Complete API Coverage** – creates granular tests to cover a company's entire API footprint, addressing all API attack breach categories. It covers all of the nooks, crannies, edge cases, and corners that are easy to miss with manual efforts.

4. **Speed of DevSecOps** – enables API testing on every nightly build or release, without adding any delay. Testing is fast, running in minutes rather than adding hours or days to a company's integration and deployment workflow.

5. **Continuous Security** – provides continuous API protection, unlike manual pen-testing that is run monthly or at longer intervals.

6. **Cost efficiency** – provides far greater test coverage of manual pen-testing at much lower cost, leveraging the efficiencies inherent in a scalable and automated process.

After completing the vulnerability analysis, APIsec integrates results into the CI/CD pipeline, creates trouble tickets in popular systems like Github and Jira, produces pen-test reports suitable for submission to FedRAMP auditors and compliance officers, and provides an easy-to-use dashboard for visualizing and managing changes to an API over time. The dashboard allows replaying the attack to show the exact nature of the vulnerability, and automatically calculates Common Vulnerability Scoring System (CVSS) severity scores.

Please visit www.apisec.ai to learn more.

Dan Barahona, Head of Marketing and Business Development
Email: dan@apisec.ai