LEARNING MADE EASY

Noname Security Special Edition

API Security Testing

for dummies®
A Wiley Brand

- Identify the top API vulnerabilities
- Understand limitations in current tools
- Adopt a shift-left testing approach

Brought to you by

noname

Lawrence Miller

About Noname Security

Noname Security is the only company taking a complete, proactive approach to API security. Noname works with 20 percent of the Fortune 500 and covers the entire API security scope — discovery, posture management, runtime protection, and security testing. Noname Security is privately held and remote-first, with headquarters in Silicon Valley, California, and offices in Tel Aviv and Amsterdam.

API Security Testing

Noname Security Special Edition

by Lawrence Miller, CISSP

for dummies®
A Wiley Brand

API Security Testing For Dummies®, Noname Security Special Edition

Published by
John Wiley & Sons, Inc.
111 River St.
Hoboken, NJ 07030-5774
www.wiley.com

ISBN 978-1-119-86976-4 (pbk); ISBN 978-1-119-86977-1 (ebk)

For general information on our other products and services, or how to create a custom *For Dummies* book for your business or organization, please contact our Business Development Department in the U.S. at 877-409-4177, contact info@dummies.biz, or visit www.wiley.com/go/custompub. For information about licensing the *For Dummies* brand for products or services, contact BrandedRights&Licenses@Wiley.com.

Publisher's Acknowledgments

Some of the people who helped bring this book to market include the following:

Project Editor: Elizabeth Kuball
Acquisitions Editor: Ashley Coffey
Editorial Manager: Rev Mengle
Senior Client Account Manager: Matt Cox
Production Editor: Mohammed Zafar

Table of Contents

Introduction

Digital transformation initiatives have forced application developers to move at an accelerated pace to meet enterprise goals and maintain market competitiveness. However, this "death march" frequently leads to DevOps teams cutting corners to execute plans and meet deadlines. Oftentimes, code quality suffers and security vulnerabilities are exposed.

Many organizations fail to adequately test their application programming interfaces (APIs). Even organizations that do test their APIs often test only for functionality, not security. API security testing is either bypassed altogether or done manually via homegrown processes without a true API security testing tool. This increases the likelihood of APIs being released with design flaws and misconfigurations.

To protect their applications and data from increasingly sophisticated threats that target API vulnerabilities and risks, application developers and DevOps teams must adopt a "shift-left" approach, testing early and often in their software development life cycle (SDLC) with an enterprise-grade API security testing solution.

About This Book

API Security Testing For Dummies, Noname Security Custom Edition, consists of five chapters that explore the following:

- What APIs are and how they're used (Chapter 1)
- The growing API attack surface (Chapter 2)
- The limitations of existing API security testing tools and approaches (Chapter 3)
- How Active Testing enables a shift-left approach to security (Chapter 4)
- Important keys to secure application and API delivery (Chapter 5)

Each chapter is written to stand on its own, so if you see a topic that piques your interest, feel free to jump ahead to that chapter. You can read this book in any order that suits you.

Foolish Assumptions

It has been said that most assumptions have outlived their uselessness, but I assume a few things nonetheless!

Mainly, I assume that you are a DevOps manager, an application security professional, or even an application developer, interested in building a meaningful DevSecOps process for your organization. As such, you recognize the need to overcome the common perception of security as a hindrance to innovation and to proactively collaborate with your application security teams to integrate effective security measures into your delivery pipelines.

If any of these assumptions describes you, then this is the book for you! If none of these assumptions describes you, keep reading anyway — it's a great book, and after reading it, you'll know quite a bit about API security testing.

Icons Used in This Book

Throughout this book, I use special icons to call attention to important information. Here's what to expect:

REMEMBER

This icon points out important information you should commit to your nonvolatile memory, your gray matter, or your noggin.

TECHNICAL STUFF

This icon explains the jargon beneath the jargon and is the stuff legends — well, legendary nerds — are made of.

TIP

Tips are appreciated but never expected, and I sure hope you'll appreciate these useful nuggets of information.

WARNING

These alerts point out the stuff your mother warned you about. Well, probably not, but they do offer practical advice.

Beyond the Book

There's only so much I can cover in this short book, so if you find yourself at the end of it wondering, "Where can I learn more?," head to `https://nonamesecurity.com`.

IN THIS CHAPTER

- Defining application programming interfaces (APIs)
- Leveraging the power of APIs in the private and public sectors

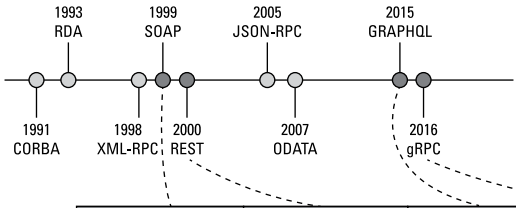
Chapter 1
Understanding Application Programming Interfaces

This chapter starts with the basics: what APIs are, what they do, and how they enable our digital world in both the private and public sectors.

What Is an API?

Application programming interfaces, or APIs, help make applications and digital services easier to consume. APIs also make it easier for developers to build, enhance, and maintain applications. How exactly? In a nutshell, APIs are software interfaces that dictate how software components interact with each other and define how data is shared and modified.

APIs can be written in practically any programming language (such as Java, Go, or C#), and many API standards exist that use Extensible Markup Language (XML), JavaScript Object Notation (JSON), and so on as a data protocol, making it possible to seamlessly transmit data between disparate systems (see Figure 1-1).



	SOAP (Simple Object Access Protocol)	REST (Representational State Transfer)	GraphQL	RPC (Remote Procedure Call)
Organized in terms of	enveloped message structure	compliance with six architectural constraints	schema and type system	local procedure call
Format	XML only	XML, JSON, HTML, plain text	JSON	JSON, XML, Protobuf, Thrift, FlatBuffers
Learning curve	Difficult	Easy	Medium	Easy
Community	Small	Large	Growing	Large
Use cases	– payment gateways – identity management – CRM solutions – financial and telecommunication services – legacy system support	– public APIs – simple resource-driven apps	– mobile APIs – complex systems – micro-services	– command and action-oriented APIs – high performance communication in massive micro-services systems

Source: Altexsoft

FIGURE 1-1: API architectural styles comparison.

APIs do two things (see Figure 1-2):

- They allow people to build applications (software) that communicate with existing applications and services.
- They allow people to build applications that perform certain actions on data.

Some APIs even give software the ability to interact with physical devices using specialized protocols.

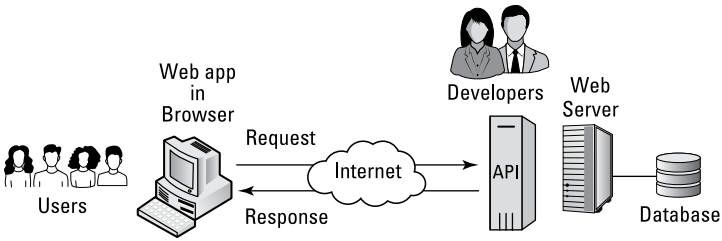


FIGURE 1-2: APIs allow applications to communicate with other applications and services.


TIP

The Twitter API is a real-world example of an API that allows people to build applications that are able to pull Twitter data and compose tweets. Using an API, it's possible for an application like Tweetbot to pull tweets from a Twitter account and allow the user to compose a new tweet without explicitly logging in to the Twitter website as a user. Similarly, the API for Gmail allows people to build applications that let users compose and send emails without ever logging in to Gmail.

With APIs, you can build applications that automatically update without requiring any manual work. You can also empower users to interact with existing applications and services in a more efficient way. This increases developer productivity by allowing them to focus on the functionality of their applications rather than on the different software components.

There are different types of APIs, some of which are used for communication between microservices. These types include Simple Object Access Protocol (SOAP), Representational State Transfer (REST), and Graph Query Language (GQL) APIs. Some APIs are intended to manipulate data, such as create, read, update, delete (CRUD) APIs.


WARNING

APIs and microservices often get confused because microservices use APIs. However, APIs are usually the communication medium between microservices, which are groups of software components that communicate autonomously. Microservices are capable of processing requests on their own, usually without requiring human intervention. Those requests can be for actions such as reading data, updating data, or even deleting data. So again, microservices leverage APIs but are not APIs themselves (see Figure 1-3).

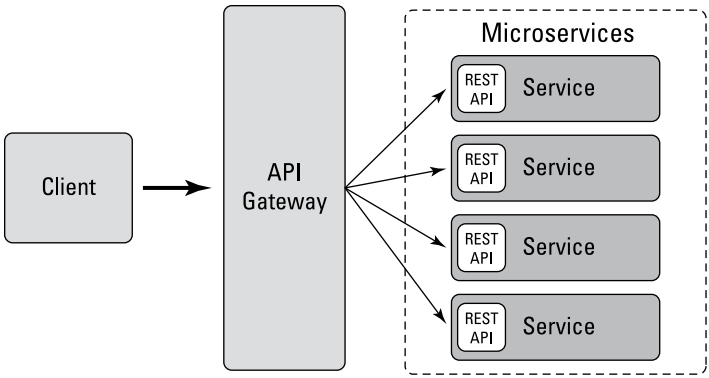


FIGURE 1-3: APIs and microservices.

How Are APIs Used in the Private and Public Sector?

Simply put, APIs connect the world, from revenue-generating customer experiences to cost-saving back-end integrations, and everything in between. Quite frankly, APIs are an integral part of practically every digital activity — in both the private and public sectors.

In the private sector, APIs enable financial institutions to enhance the consumer experience and streamline payments. APIs enable retailers to generate new revenue streams, optimize existing processes, and strengthen relationships with customers. Likewise, healthcare organizations use APIs to seamlessly share data across the continuum of care.

WARNING

According to Marsh McLennan Global Cyber Risk Analytics Center, "healthcare API traffic grew by more than 400 percent in 2020, and health monitoring API use increased an additional 941 percent in 2021." This explosive growth in API usage presents a rich target for cybercriminals to exploit sensitive data. And the reality is, healthcare organizations can't afford to sit idle — especially when considering the average cost of a data breach in healthcare reached $9.42 million per incident, according to IBM Security.

The impact of APIs can be seen across all areas of the public sector, including education, transportation, healthcare, social services, and law enforcement. APIs enable government agencies to seamlessly share data across federal, state, and local levels. APIs

are also essential for ushering in advanced capabilities and new functionality for citizens, veterans, and government personnel.

However, despite the myriad benefits that APIs present to government agencies, APIs also expand the attack surface. And with API usage skyrocketing, agencies need to proactively guard against the new security risks that APIs present. As the number of APIs quickly surpasses the manual span of control and capabilities of existing legacy security controls, public agencies are facing increasing challenges when it comes to security. To put things into perspective, the average cost of a data breach in the public sector has surged 78.7 percent year over year to $1.93 million per incident, according to IBM Security.

WARNING

Unfortunately, many government agencies look at APIs as part of traditional application security. The reality is, AppSec and DevOps personnel need to think about APIs separately, with their own security considerations. APIs present their own unique risks, which legacy tools can't address. Agencies need to partner with the right API security vendor in order to build a complete governance and security program for their APIs.

CUSTOMER SUCCESS STORY: RAPYD

Rapyd is the fastest way to power local payments anywhere in the world, enabling companies across the globe to access markets quicker than ever before. By utilizing Rapyd's unparalleled payments network and Fintech as a Service (FaaS) platform, businesses and consumers can engage in local and cross-border transactions in any market. The Rapyd platform is unifying fragmented payment systems worldwide by bringing together 900-plus payment methods in more than 100 countries.

Challenges

Rapyd's main product is its public payments API, which handles billions of dollars of transactions 24/7. Even minor instances of

(continued)

(continued)

disruptions, fraud, or abuse could mean millions of dollars in lost revenue, significant remediation costs, and a loss of customer trust for both Rapyd and its customers.

Although Rapyd ran an active bug bounty program, significantly customized its web application firewall (WAF), and considered API security mission-critical, its APIs were a "black box" to its security team. It lacked granular visibility into API usage and behavior, business logic was unknown, and it was difficult to identify — let alone stop — attacks in real time.

Consequently, Rapyd's security team needed a better way to secure both its public API and its hundreds of internal APIs in a highly complex system operating in Amazon Web Services (AWS) at a global scale. This meant a purpose-built API security solution that didn't have the blind spots of its existing infrastructure, including WAFs and API gateways. Rapyd needed a granular inventory of all its APIs, visibility into mistakes or misconfigurations creating vulnerabilities in its security posture, intelligently prioritized alerts so security analysts could focus on the most important risks, and the automation and integrations necessary to stop attacks.

Solution

Rapyd's chief information security officer (CISO) evaluated a number of established purpose-built API security solutions, including from vendors with numerous patents and long track records. However, most fell short of providing complete API security because they lacked important capabilities, such as full packet capture for deep analysis of attacker behavior, visibility beyond traffic and anomalies into its global API security posture, and the backing of world-class security researchers.

Unlike other vendors and the "API security" features of their current infrastructure, only Noname Security provided the combination of comprehensive visibility from code to production, discoverability, automation, integrations, and intelligent behavior-based anomaly detection that Rapyd needed.

From their first meeting, Noname Security demonstrated an intense customer focus, level of expertise, and industry leadership that surpassed other vendors.

After evaluating each vendor's holistic combination of product and team capabilities, Noname Security emerged as the clear leader. The CISO's team quickly deployed the Noname API Security Platform — with posture management, runtime protection, and Active Testing in one unified solution — across all its AWS regions globally.

Results

With the Noname API Security Platform, Rapyd can protect its APIs and critical assets from cyberattacks with:

- Easy, effective, and accurate API behavioral prevention, detection, and response
- Effective resource utilization to proactively de-risk the environment
- Evidence of security control and demonstration of compliance
- Secure handling of sensitive data and third-party risk exposure

Rapyd can now confidently grow its global business both quickly and securely, as real data from blocked attacks and production vulnerabilities inform its development efforts and new code can be easily tested before going live. Rapyd will also have full architectural freedom to deploy Noname as fully cloud-based, fully on-premises, or any hybrid combination as needed as it continues to expand into new markets and regulatory environments.

IN THIS CHAPTER

- Surveying the burgeoning application programming interface (API) attack surface
- Looking at the top API threats
- Addressing other API vulnerabilities

Chapter 2
Exploring API Risks and Vulnerabilities

This chapter explores the rapidly growing application programming interface (API) attack surface, the top threats to APIs, and other vulnerabilities that need to be addressed in a robust API security program.

Looking at the Growing API Attack Surface

APIs are the backbone of today's app-driven world, and they establish key conduits for working both inside the enterprise and with partners and customers. But these critical byways are under attack, and simple API misconfigurations have already led to major breaches.

APIs are specifically designed to allow access to software services that may contain sensitive data. Unfortunately, many organization's API ecosystems are fraught with vulnerabilities. Too often, enterprises only become aware after a breach has already occurred. API security is complex, and even for organizations that are proactively managing API risk, the range of vulnerabilities and security risks associated with APIs can be daunting.

WARNING

According to *The API Security Disconnect – API Security Trends in 2022*, 76 percent of respondents suffered an API security incident in the last 12 months.

Identifying the Top Threats

The Open Web Application Security Project (OWASP) API Security Top 10 is an excellent starting point to help organizations identify the most critical threats to their API footprint.

OWASP classifies each API security threat according to four criteria:

- Exploitability
- Weakness prevalence
- Weakness detectability
- Technical impact

Each factor is given a score, with three being the most severe (see Figure 2-1). A vulnerability that is easy to exploit, widespread, and easily detectable with severe technical impact is the most urgent to address. These dimensions allow API security risks to be force-ranked in terms of severity.

Threats Agent	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impact
API Specific	Easy 3	Widspread 3	Easy 3	Severe 3
	Average 2	Common 3	Average 3	Moderate 3
	Difficult 2	Difficult 3	Difficult 3	Minor 3

FIGURE 2-1: OWASP scoring criteria.

The OWASP API Security Top 10 (2019) includes the following vulnerabilities (note that OWASP is typically updated every three years, and the 2022 Top 10 is currently in development):

- **Broken object level authorization:** APIs often expose endpoints that handle *object identifiers* (that is, a unique

value that describes an object), resulting in a wide attack surface in access level control. As a result, attackers may be able to read, update, delete, or create data objects without permission by exploiting broken object-level authorization vulnerabilities. Thus, an attacker may be able to forge a ticket that grants them unauthorized permissions, analogous to forging a permission slip to go on a field trip in school. Individual authorization checks are necessary for every function that interacts with a data source.


TECHNICAL STUFF

Authentication verifies who you are (your identity). *Authorization* verifies what you're allowed to do (your permissions).

- **Broken user authentication:** Authentication is the process of verifying that an individual, entity, or website is who it claims to be. As part of the authentication process of web applications, a user is usually required to submit a username (or ID) and one or more items of private information that are known only to the user. Authentication is more complex to implement for APIs because multifactor authentication (MFA) and other human-driven authentication options aren't possible. Also, authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to take over other users' identities temporarily or even permanently. Thus, an attacker can impersonate an authorized user or object to gain access to an API. Compromising the ability to identify the user compromises overall API security.
- **Excessive data exposure:** Sometimes, in an effort to be able to do a lot, APIs say too much. To promote utility, developers tend to expose all object properties without considering their individual sensitivity, depending on clients to filter the data before showing it to the user.
- **Lack of resource and rate limiting:** In many cases, APIs do not restrict the size or number of resources that can be requested by the client in a session or a certain period of time (that is, rate limiting). This means users can make huge requests, either inadvertently or maliciously. This can not only negatively impact API server performance, leading to denial of service (DoS) attacks, but also leave the door open to authentication flaws such as brute-force attacks.
- **Broken function-level authorization:** Authorization flaws tend to occur in complex access control policies with

hierarchies, roles, and groups, and an unclear distinction between administrative and regular functions. An attacker can gain access to other users' resources or admin functions by exploiting these weaknesses.

- **Mass assignment (also known as autobinding and object injection):** Various software frameworks allow developers to automatically bind Hypertext Transfer Protocol (HTTP) request parameters to objects or variables for ease of use. However, this creates an opportunity for attackers, who might modify object properties by guessing objects' properties, exploring API endpoints, reading documentation, or including additional object properties in request payloads. Client-provided data (for example, JavaScript Object Notation, or JSON) bound to data models without proper properties filtering based on an allow list usually results in mass assignment.
- **Security misconfiguration:** Security misconfiguration is just that — a misconfiguration in the security parameters of the API, often found in relation to open cloud storage, insecure default configurations, incomplete or ad-hoc configurations, misconfigured HTTP headers, unnecessary HTTP methods, permissive cross-origin resource sharing (CORS), and/or verbose error messages containing sensitive information.
- **Injection:** If data and code aren't carefully separated, problems arise. When untrusted data is sent to an interpreter as part of a query or command, injection flaws — such as Structured Query Language (SQL) and NoSQL — command injection can occur. An attacker can manipulate malicious data to trick the interpreter into performing unintended commands or accessing information without authorization.
- **Improper assets management:** Good API inventory and documentation are essential because APIs expose more endpoints than traditional web applications do.
- **Insufficient logging and monitoring:** Most APIs weren't inherently designed to be good at reporting into top-level security tools. When insufficient logging and monitoring is coupled with ineffective integration to incident response, hackers can further attack systems, maintain persistence, and pivot to other systems to extract, tamper with, or destroy data. It often takes more than 200 days for a breach to be detected, and most breaches are detected by external parties rather than by internal processes or monitoring.

TIP

Download the Mitigating OWASP Top 10 API Security Threats e-book at `https://nonamesecurity.com` to learn more about the OWASP Top 10 and how to remediate these vulnerabilities.

Addressing Other API Vulnerabilities

The OWASP API Security Top 10 is composed of the top ten vulnerabilities — but there are many more vulnerabilities that organizations need to test for and remediate in their APIs. In fact, a "less severe" vulnerability that is actively being exploited by a threat actor can be a far greater risk to your organization than a "more severe" vulnerability that is not being actively exploited.

API vulnerabilities can be broadly classified into one of three categories:

- **Mistakes:** API security flaws are often the result of simple human error. APIs can be just as complex to develop and maintain as the applications they connect to. Thus, it's easy to misconfigure policies, to host multiple versions of an API, and for temporary APIs to be left open by time-pressed teams.
- **Misfortune:** An unforeseen sequence of events can expose an API to the open web or allow unauthorized access. As the use of APIs explodes, such mishaps will likely occur with increasing frequency. All companies know that things will go wrong — and it's incumbent upon them to quickly detect those failures and close the security gaps they create.
- **Malice:** The third type of vulnerability comes from a deliberate attack on the API itself. It could originate from outside the organization or from within. By their nature, APIs are well documented and version controlled. It's evident how the API should respond, so a hacker can look to exploit it armed with that information. For instance, malicious code designed to subvert or disrupt the service can be injected into a feature allowing a user to upload a profile photo. An example of this is the broken object-level authorization (BOLA) vulnerability in Coinbase's retail advanced trading API, discovered by a member of the crypto community in early 2022. Or an authentication challenge may be attacked through credential stuffing overwhelming the service. Some

versions of standard APIs are vulnerable to specific attack vectors, and hackers will probe them to see whether those exploits are live before launching their attacks. In addition, bad actors are looking to exploit deployment and configuration mistakes made by organizations and take advantage of their misfortune when the opportunity strikes.


TIP

Securing APIs is complex and requires an approach that encompasses all aspects of the API from its development, deployment configuration, and runtime operations. Here are the four key areas organizations need to consider:

- **API discovery:** API discovery tools help you locate every API in your environment, including legacy and shadow APIs. Based on that inventory, determine which APIs are critical to the business.
- **API security posture:** Successful API posture management depends on your ability to assess their configurations for misconfigurations and vulnerabilities. Ensure your API security platform has data classification capabilities so you can identify how many APIs are able to access credit card data, phone numbers, Social Security numbers (SSNs), and other sensitive data.
- **Detection and response:** Deploy behavioral-based models for runtime API threat detection. You should use a traffic monitoring platform that runs out of band to avoid any negative impact on performance. Implement automated and semiautomated for blocking and remediation of threats.
- **Continuous testing:** Continuously test API endpoints to identify API risks before they emerge. Supplement DevOps static application security testing (SAST), dynamic application security testing (DAST), software composition analysis (SCA), and other existing tools (all discussed in Chapter 3) with API-specific testing that can be automated and incorporated into continuous integration (CI)/continuous delivery (CD) pipelines.

IN THIS CHAPTER

- Analyzing source code with static application security testing
- Probing running applications with dynamic application security testing
- Taking a hybrid approach with interactive application security testing
- Scanning open-source software components
- Looking at the need for application programming interface security testing

Chapter 3
Recognizing the Limitations of Existing Tools and Approaches

This chapter explores existing application security testing tools and approaches and explains why these tools aren't sufficient for application programming interface (API) security testing.

Static Application Security Testing

Static application security testing (SAST), sometimes referred to as source code analysis or static analysis, is a white-box methodology for testing that analyzes application source code for security vulnerabilities before it's compiled. In SAST, testers inspect an application's inner workings, inspecting noncompiled source code to identify syntax errors, input validation problems, and so forth. Using specialized tools, it's also possible to do SAST on compiled code.

TECHNICAL STUFF

In white-box testing, the tester knows the internal structure, design, and implementation of the component being tested.

SAST is a commonly used application security tool that identifies and helps remediate the root causes underlying security vulnerabilities. SAST tools don't need a system to be running to perform a scan because they analyze web applications from the inside out. For example, SAST testing may be used for regulatory compliance with the Payment Card Industry Data Security Standard (PCI DSS), or to improve insight into software risk.

SAST tools can analyze an entire code base much more rapidly than humans can. It takes only minutes for SAST tools to scan millions of lines of code to identify critical vulnerabilities. Ultimately, these tools help organizations achieve key goals, such as the following:

- **Secure coding:** SAST readily identifies basic coding errors so development teams can easily comply with best practices for secure coding.
- **Ease of use:** Integrating SAST earlier in continuous integration (CI)/continuous delivery (CD) pipelines and DevOps environment saves developers from needing to trigger or separately configure scans. This makes it more efficient, more convenient, and easier for them to use, and eliminates the need for developers to leave their environment to conduct scans, view results, and remediate security issues.

However, there are some important limitations of SAST, including the following:

- **False positives:** Vulnerabilities may be identified that can't actually be exploited (for example, because of input validation).
- **Language dependent:** Most SAST tools only support one or perhaps a few languages and frameworks, so you may need multiple SAST tools for different APIs.

REMEMBER

Because it can take place without code being executed and doesn't require a working application, SAST takes place very early in the software development life cycle (SDLC). This helps developers quickly resolve issues and identify vulnerabilities in the project's initial stages without passing vulnerabilities on to the final application.

Dynamic Application Security Testing

Dynamic application security testing (DAST) is an advanced testing method for an application in an operating state. The process focuses on testing the production environment and analyzing application security at runtime. It tests how systems and components interact in practice and identifies real-world vulnerabilities without much need for insight into the provenance of any single component.

DAST testing is operational and behavioral in that testers identify problems that occur during use and then trace them back to their origins in the software design, instead of detecting issues linked to a code module. It's useful for basic security on evolving projects and for achieving industry-standard compliance.

DAST benefits application security as a whole in many ways. One of the primary benefits is that a DAST security tester tries to hack an application as an attacker would while the application is running. Some additional benefits include the following:

- **Technology independent:** Because it doesn't rely upon source code, DAST is language and platform agnostic. Not being limited by particular technologies and languages allows organizations to run a single DAST tool on all their applications.
- **Fewer false positives:** According to the Open Web Application Security Project (OWASP) Benchmark Project, there is a lower false-positive rate from DAST and less noise than with other application security testing tools.
- **More configuration issues identified:** Because DAST focuses on detecting operational security vulnerabilities and attacks applications from the outside in, it's well suited to identifying configuration mistakes that other application security testing tools miss.

Although there are many benefits to using DAST, there are also a number of limitations that would encourage developers to seek other means of testing, including the following:

- **Lack of scalability:** DAST relies heavily on effective tests, and security experts are needed to write them. This makes scaling DAST very difficult because there are often a limited number of expert resources available.

- **No API discovery:** DAST can't discover APIs, so you're missing asset management and API reachability validation capabilities.
- **Minimal visibility:** DAST lacks visibility into the application's code base, so DAST alone can't offer comprehensive security coverage or insight into problematic code for purposes of remediation.
- **Time-consuming:** DAST can be slow — according to Forrester, DAST scans can last as long as five to seven days. DAST scans often don't detect vulnerabilities until they're more costly and time-consuming to fix, later in the SDLC.

Although many people see DAST as an always-automated approach, it's widely divided into two types:

- **Manual DAST** simply refers to using knowledge of a specific field and experience to detect vulnerabilities that DAST scanners might miss.
- **Automated DAST** includes feeding data to DAST protection software to test applications. This type of automated test includes scanners, fuzzers, crawlers, and other tools that can identify vulnerabilities such as cross-site scripting, SQL injection, and server-side request forgery. For example, a DAST attack can send a large string of numbers to help identify a SQL injection flaw.


TECHNICAL STUFF

Fuzzing (or fuzz testing) submits bad or malformed input into the fields within an application to determine how the application will respond (for example, entering an unknown state or crashing). The fuzzing process is shown in Figure 3-1.

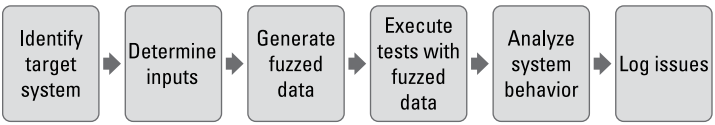


FIGURE 3-1: The fuzzing process.


TIP

The best DAST software and other DAST tools simulate various types of attacks to detect security vulnerabilities and test a broad spectrum of endpoints, including hidden values. By simulating malicious attacks on an application, automated DAST security tools can help identify outcomes that are far outside the typical user experience.

DAST products function without getting into the source code, so they require no prior knowledge of programming language. This makes DAST software easy to use, and because DAST detects vulnerabilities at runtime, there is no need to rebuild an application to test it for vulnerabilities.

REMEMBER

Although they may seem similar, there are important differences between DAST and penetration testing (or pentesting). DAST systematically focuses on the running state of the application, while pentesting (with owner permission) uses common attack techniques to exploit vulnerabilities in the application and beyond, including ports, firewalls, servers, and routers. During pentesting, simulated attacks are launched to find computer system vulnerabilities and identify weak spots that attackers could exploit. Modern pentesting blends technology and automation with human expertise.

Interactive Application Security Testing

Like DAST, interactive application security testing (IAST) focuses on application behavior during runtime. However, IAST analysis takes more of a hybrid approach, combining analysis of internal application flows with scanning and black-box testing. IAST is most beneficial in its ability to connect source code with DAST-like findings. But this also makes IAST programming-language dependent (because it needs to scan source code) and restricts it to being performed later in the CI/CD pipeline.

TECHNICAL STUFF

In black-box testing, the tester doesn't know the internal structure, design, and implementation of the component being tested.

Software Composition Analysis

With the world of software becoming increasingly complex and interconnected, it's more important than ever before to scan a code base for vulnerabilities before using it or integrating it into your own software.

Software composition analysis (SCA) is a subset of application security testing that refers to an automated process that scans

open-source software, allowing security analysts to identify precisely which libraries and components have been used in a piece of software. Code is parsed automatically and scanned against a known list of open-source vulnerabilities.

Specifically, SCA tools can be used to inspect

- Raw source code
- Containers
- Binaries
- Components of an operating system

SCA tools can be stand-alone or can be integrated into wider-ranging application testing solutions. For instance, one SCA tool may be used to identify what's running on Linux workstations, while another could be used to inspect finished binaries as they move through the DevOps process. They're also commonly used to create a bill of materials (BOM) for software developers. BOMs will list all the components used in a piece of software, allowing the security team to flag any potential vulnerabilities.

SCA tools can also be used to identify the licenses associated with any open-source components integrated into a piece of delivered software.

REMEMBER

Open source doesn't necessarily mean free. By scanning code bases for licenses with SCA tools, developers can avoid costly fines and reputational damage.

Organizations that use an SCA tool in their development workflow can realize several benefits, including the following:

- **Vulnerability management:** The primary rationale for using technologies leveraging SCA is avoiding forking code that has known vulnerabilities. Using SCA tools, these can be quickly identified and, where possible, remediated. If a bug fix isn't available or can't be deployed quickly, the development team may decide to forgo using a vulnerable component altogether.
- **Bill of materials:** One of the main SCA benefits is the fact that by cataloging which open-source components are used in a given piece of software it can be used to help collate a

software BOM. Sometimes, producing a software BOM is a regulatory requirement — or a requirement that is defined by potential customers.

- **Licensing:** Your organization can avoid costly fines that could be incurred by accidentally using open-source components that require licenses that the organization hasn't acquired.

SCA supports API security by providing a code-level security assessment. In order to secure APIs and their endpoints, it can be helpful to know more about the services that are attempting to connect with them. This can be used, for example, to help identify malicious software or software with unresolved open-source vulnerabilities flooding endpoints with requests.

Therefore, SCA tools can provide a useful component in a holistic strategy intended to protect APIs through preventing their unwanted use, degrading service availability for legitimate users.

REMEMBER

SCA tools may sound a lot like SAST tools. However, they're two distinct cybersecurity components with slightly different uses. Some of the differences that mark the divide between SAST and SCA:

- SAST tools are used to detect vulnerabilities in code that is closed source (that is, proprietary). By contrast, SCA tools specialize in flagging vulnerabilities in open-source components.
- When it comes to prioritizing bug fixes, SCA tools result in fewer developer headaches because remediating open-source vulnerabilities is relatively straightforward given that the overwhelming majority of known bugs have a fix. By contrast, remediating bugs detected in closed-source components can be more arduous work.
- Whereas SCA tools can run on software in a variety of formats — both before and after compilation — SAST tools typically run only on source code.

Recognizing the Need for API Security Testing

The problem with SAST, DAST, IAST, and SCA is that these tools don't do much for API security testing. At least, they can't do it directly.

An API security problem may manifest itself in SAST or DAST test results, but the root issue may not be evident. For example, a DAST process could reveal a problem with query strings, but the actual risk may be coming from a compromised API run by another entity. To figure that out, the testing team has to investigate and hopefully find that API. They may spend a lot of valuable time — time they don't have — looking for the API and may never find it.

Another example of an API security risk that won't get picked up by existing application security testing approaches is known as *fuzzing*. A fuzzing attack involves a malicious actor using "fuzzy" approximation of the data required for an API call to see if they can get a response or break something. For example, if the API is expecting a user ID, the attacker can send a large number of randomly generated IDs hoping to stumble upon a real user ID in the process. If there are no controls restricting this type of attack, then the API and the data it fronts may be at risk of a breach.


TIP

Chapter 4 explains more about the value of API security testing and how Active Testing enables a shift-left approach to API security testing.

IN THIS CHAPTER

- Testing your application programming interfaces (APIs) early and often
- Introducing Active Testing
- Realizing the benefits of API testing

Chapter 4
Adopting a Shift-Left Approach

This chapter explores the importance of testing APIs early and often. You learn about Noname Security's Active Testing solution, and you discover the benefits of adopting a shift-left approach.

Understanding the Need for API Security Testing

As the number of APIs rapidly skyrockets, companies are facing growing challenges when it comes to security, including the following:

- There aren't enough security personnel who know how to test APIs.
- The number of APIs is growing faster than the security team can keep up with.
- Existing security tools lack adequate coverage.

Any one of these challenges can spell disaster for your environment. However, there is one overlooked aspect that could also weaken your API security posture if not addressed, and that's testing APIs early in the development process.

When it comes to API development, it's not just a matter of *if* you test, but also *when* you test your APIs. The traditional model places testing closer to the deployment phase. Testing at this stage is definitely a vital step, but testing *only* at this stage is insufficient and can lead to serious vulnerabilities. How exactly? By consolidating testing into one phase of the software development life cycle (SDLC), you create a bottleneck in the process with a mountain of code to test.

To alleviate this bottleneck and expedite the process, certain steps in this testing phase may not be fully completed or may be bypassed altogether. As a result, code quality suffers — and your API attack surface grows. And because this is the one phase in the process for testing, anything missed here won't be caught until it's too late.

Shift left (see Figure 4-1) is an approach of moving certain tasks to an earlier stage in the development process. This means that tasks that are traditionally done at a later stage should instead be performed at earlier stages — particularly those tasks related to API security and software testing.

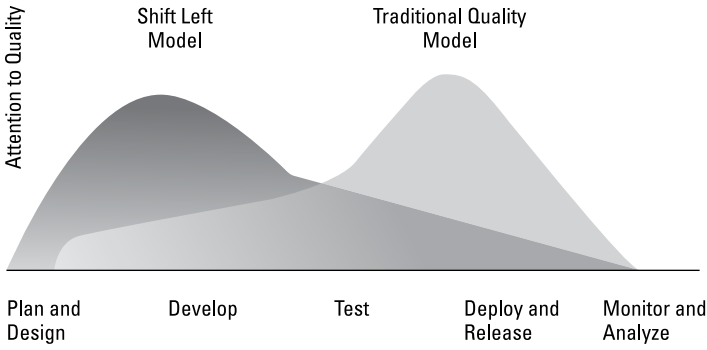


FIGURE 4-1: In the shift-left model, API security and software testing tasks are performed earlier in the SDLC.

With security and testing baked into each step of the API development or DevOps process, a shift-left approach ensures developers will be identifying and remediating vulnerabilities throughout the SDLC. Shift-left principles enable security teams to increase developer autonomy by providing support, expertise, and tooling while still delivering the required level of oversight. Developers can release more secure code at scale, build API security into their designs, and make fixes early in the development process instead of scrambling to fix them later. Code testers are able to evaluate features as they're created and help ensure higher quality.

The shift-left testing process is continuous, running parallel with development, and involves continuous communication between customers, developers, and testers. The shift-left testing process typically involves several steps:

1. Studying customer requirements, application behavior, and end-user expectations
2. Developing tests for unit, integration, and functionality
3. Executing tests via end-to-end automation
4. Running non–user interface (UI) tests as they're implemented

Shift-left testing also helps minimize defects along the way by encouraging both test-driven development (TDD) and behavior-driven development (BDD).

REMEMBER

Shift-left testing saves time by identifying bugs earlier in the life cycle. More rigorous detection of errors and bottlenecks in advance enables testers to improve initial designs and develop alternatives. The shift-left approach ensures quality by allowing the development team to bake automation tools like API testing or unit testing right into the building process. Ultimately, apps, microservices, and APIs are more sufficiently protected. Waiting to test until production means the team is always catching up and dealing with inherently greater risk.

Introducing Active Testing

Noname Security Active Testing helps customers stop vulnerabilities before production and innovate faster. True to a shift-left approach, security testing is baked into each step of the process,

ensuring developers can monitor for vulnerabilities throughout the life cycle. Developers can build API security into the design and make fixes early.

Active Testing focuses on integrating with existing process that already test code, delivering security insights, providing root-cause remediation insight, and pausing the build pipeline (if, and when appropriate) to avoid releasing vulnerable code into production.

Active Testing enables organizations to

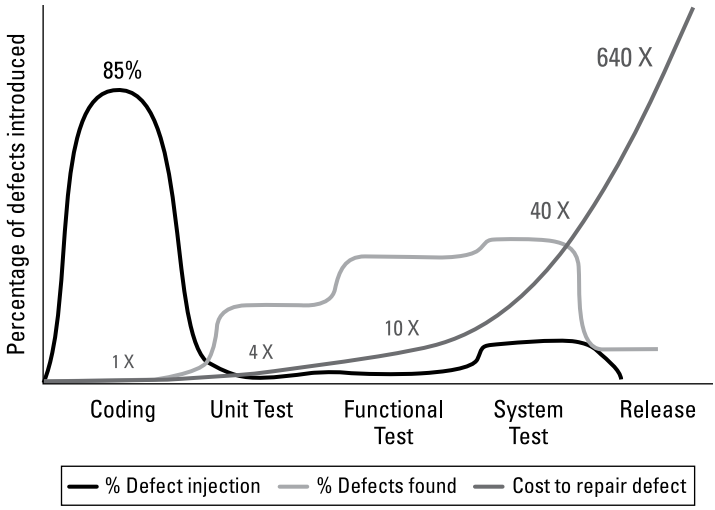
- **Stop vulnerabilities before production.** True to the shift-left approach, Noname Active Testing provides a suite of API-focused security tests that security operations (SecOps) teams can run on demand, or as part of a continuous integration (CI)/continuous delivery (CD) pipeline to ensure that APIs aren't implemented with security vulnerabilities in them. Active Testing automatically runs more than 150 dynamic tests that simulate malicious traffic, including against the Open Web Application Security Project (OWASP) API Top 10.
- **Innovate faster.** Developers can release more secure code at scale, build API security into the design, and make fixes early in the development process instead of scrambling to fix them later. Code testers are able to evaluate features as they're created and help ensure higher quality. Ultimately, security teams increase developer autonomy by providing support, expertise, and tooling while still delivering the required level of oversight.


TIP

Noname Security customers see Active Testing as a strategic benefit that allows them to replace a subset of their existing application security testing tools in favor of a purpose-built API security testing solution.

Realizing the Value of Testing Early and Often

Testing early and often saves time and money by helping to quickly identify potential security issues earlier in the API life cycle (see Figure 4-2).



Source: Jones, Capers. Applied Software Measurement: Global Analysis of Productivity and Quality.

FIGURE 4-2: Shift-left testing helps identify potential security issues earlier in the API life cycle.

Rigorous detection of errors and bottlenecks enables testers to improve initial designs and develop viable alternatives when necessary. Ultimately, applications, microservices, and APIs are better protected because organizations can proactively reduce their attack surface. Shifting left for API security helps organizations to

- **Deliver secure apps and APIs faster.** Automated API security testing enables teams to create and ship secure code without having to become security experts. By eliminating vulnerabilities early, organizations can focus on delivering the best products and services to their customers.
- **Lower costs.** Regular and frequent early testing for API security significantly reduces remediation costs, often by 10 to 100 times. Automated testing also reduces the need for costly third-party penetration testing efforts.
- **Reduce risk.** By proactively identifying and fixing vulnerabilities, organizations also reduce the potential number of threats and inadvertently exposed assets, reducing overall risk.
- **Increase revenue.** Automated testing allows developers to move quickly to meet customers' needs by ensuring that new releases are secure and unlikely to require refactoring or costly remediation in the future.

IN THIS CHAPTER

- Knowing how your business uses application programming interfaces (APIs)
- Recognizing the threats to APIs
- Defining your risk tolerance and desired security posture
- Using the right tools and processes
- Monitoring cybersecurity trends across the software development life cycle

Chapter 5
Five Keys to Rapidly Delivering Secure Applications and APIs

Here are five keys to help your organization succeed in rapidly delivering secure applications and application programming interfaces, or APIs:

- **Understand the role of APIs in your business.** Begin with a complete inventory of all your APIs. If you don't know what APIs you have, you don't know what APIs are communicating sensitive information or being exposed to the open web.

  For each API, include the following information:

  - The business and technical function of the API
  - Who the API owner is
  - Which app or business unit the API belongs to
  - Who accesses the API

- What applications and data are potentially exposed by the API and their communication flows
- What the infrastructure powering the API looks like
- The business criticality of the API

- **Understand the threats to APIs.** APIs have become mission-critical and have changed in four fundamental ways, thereby increasing their threat exposure:
  - **They're powerful.** APIs are more powerful than ever, and they're used for nearly any type of control or data exchange — from lifesaving medical devices to financial exchanges to real-time communication.
  - **They're ubiquitous.** APIs are now ubiquitous throughout the enterprise — there are tens of thousands of APIs (and growing) across multi-cloud environments, mobile applications, financial systems, and more.
  - **They're hidden.** APIs are now more hidden, embedded in business logic and executing highly complex, cross-domain functions everywhere in the enterprise.
  - **They're easily exploited.** APIs present significant vulnerabilities that are relatively easy to exploit and hard for the enterprise to identify.
- **Establish your organization's risk tolerance and desired security posture.** Risk tolerance is about understanding what's acceptable to your organization's leadership. Security posture is about knowing where you stand. Understanding your organization's risk tolerance and security posture is critical to making appropriate decisions that impact the development and maintenance of your APIs and your overall security posture.
- **Implement the processes and tools needed, as determined by your risk tolerance.** Your organization's risk tolerance will help you determine what processes and tools are needed to appropriately manage your security posture. A lower risk tolerance will require more robust and frequent testing compared to a higher risk tolerance, but it's important that your leadership fully understands the potential implications to your overall security posture given their established risk tolerance.

- **Continuously monitor trends in runtime and development to deliver secure products.** API security risks and issues aren't all discovered in source code alone. Observing traffic behavior within the context of the network provides the full context to derive risk findings. Blocking runtime API threats requires an ability to protect data by leveraging more context — information arising from an application-level understanding of how each API is intended to be used.

REMEMBER

Adopting these five keys and implementing a shift-left approach to test security early in the development process with Noname Security Active Testing will help your organization deliver secure applications and APIs faster. Visit `https://nonamesecurity.com` to learn more.

noname

The Complete API Security Platform

Deliver Secure APIs Faster

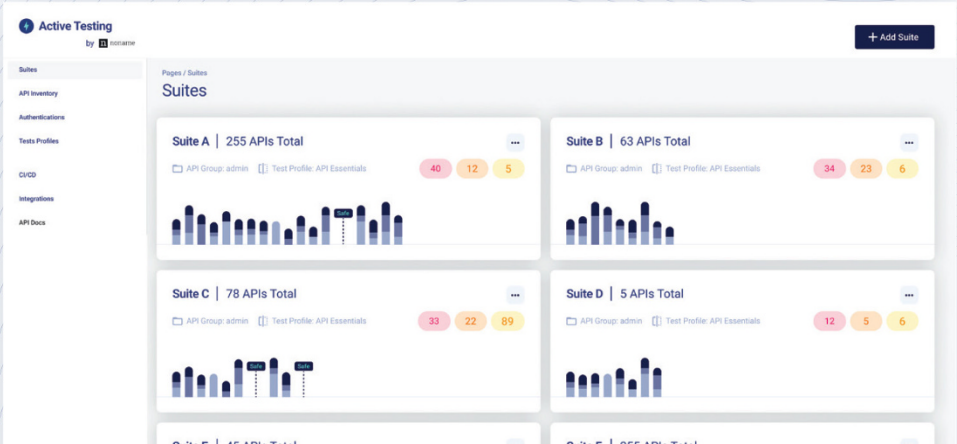
Noname Security Active Testing is a purpose-built API security testing solution that help you stop vulnerabilities before they reach production.

Test Automatically

Automatically run 150+ dynamic tests that simulate malicious traffic, including the OWASP API Top Ten.

Integrate with CI/CD

Fully integrate with your existing CI/CD pipelines, such as Jenkins and Postman, as well as all your ticketing and workflow tools such as ServiceNow, Slack, and Jira.



Deliver secure APIs faster

Application programming interfaces (APIs) are the foundation of the global digital transformation — they now account for 80 percent of total Internet traffic. Today's enterprises now manage an average of 15,000 APIs, and that number will continue to grow as thousands of new applications are introduced each year. Surprisingly, despite the fact that APIs handle some of the world's most sensitive data, organizations are struggling with API security. As a critical first step to changing this reality, organizations must test for vulnerabilities early in the software development life cycle to ensure robust API security.

Inside...

- Fix vulnerabilities before production
- Minimize risk of data leaks and breaches
- Release more secure code at scale
- Eliminate redundant pentesting
- Increase confidence in your APIs

Lawrence Miller served as a Chief Petty Officer in the U.S. Navy and has worked in information technology in various industries for more than 25 years. He is the co-author of *CISSP For Dummies* and has written more than 250 *For Dummies* books on numerous technology and security topics.

ISBN: 978-1-119-86976-4
Not For Resale

for dummies®
A Wiley Brand

WILEY END USER LICENSE AGREEMENT

Go to www.wiley.com/go/eula to access Wiley’s ebook EULA.