

Brought to you by



API Security

dummies®
A Wiley Brand

Third Salt Security Special Edition

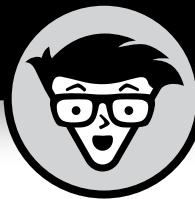


Inventory and
assess your API

Secure your APIs
throughout the lifecycle

Defend against
API attacks

Eric Schwake



API Security

Third Salt Security Special Edition

by Eric Schwake

for
dummies[®]

A Wiley Brand

API Security For Dummies®, Third Salt Security Special Edition

Published by

John Wiley & Sons, Inc.

111 River St.

Hoboken, NJ 07030-5774

www.wiley.com

Copyright © 2025 by John Wiley & Sons, Inc., Hoboken, New Jersey. All rights, including for text and data mining, AI training, and similar technologies, are reserved.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, For Dummies, the Dummies Man logo, The Dummies Way, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. Salt Security and the Salt Security logo are trademarks or registered trademarks of Salt Security, Inc. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc., is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT

ISBN 978-1-394-33840-5 (pbk); ISBN 978-1-394-33842-9 (ePub);
ISBN 978-1-394-33841-2 (ePDF)

Publisher's Acknowledgments

Development Editor: Chad R. Sievers

Acquisitions Editor: Traci Martin

Editorial Manager: Rev Mengle

Client Account Manager:

Cynthia Tweed

Production Editor:

Umeshkumar Rajasekhar

Introduction

Application programming interfaces (APIs) serve as the building blocks of modern application architecture and system design. They create the on-ramps to the digital world, keep everyone connected, facilitate business, make digital transformation possible, and continuously evolve modern computing. In all your digital activities — across employee business applications, ecommerce sites, health services, connected cars, banking applications, home automation, and mobile apps — you're using APIs. The fact that APIs enable so much sharing of data and services makes them prime targets for attackers. Security practitioners must adapt for the world of APIs so they're better equipped to secure their organization's applications and data.

About This Book

API Security For Dummies, Salt Security Custom Edition, describes how application architecture has evolved, how APIs are the foundation of modern design, how those API foundations are threatened, and how the API building blocks can be secured. This book is conveniently organized into five chapters that do the following:

- » Arrive at a working definition of modern APIs so you can dig into the many facets of API security.
- » Describe how application architecture has evolved, how cloud technologies have impacted designs, and how DevOps practices accelerate API growth.
- » Detail API implementation and operational activities that factor into security, including documentation, testing, mediation, and integration.
- » Describe how attackers abuse the business logic of APIs and automate attacks to increase likelihood of success or do further damage.
- » Increase awareness around the Open Web Application Security Project (OWASP) API Security Top 10 and the common API security gaps it spotlights.
- » Highlight the technological capabilities you need to secure APIs throughout their life cycles, including discovery, governance, and protection.

» Present ten prioritized things you can do now to start securing APIs for your organization.

Foolish Assumptions

Although this book is written primarily for technical readers with some level of experience in modern computing, everyone can benefit from the information. I make very few assumptions when writing about the world of APIs and API security. I assume you've browsed a website or used mobile application in recent years and are inclined to dig deeper into some inner workings. Knowing how your digital world functions and is built on the foundations of APIs is interesting in its own right. And it's also enlightening as to how the digital world can be threatened by malicious actors.

Icons Used in This Book

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



TIP

This icon indicates that the information is useful and can save time for a given activity.



REMEMBER

When you see this icon, make sure you read and understand the surrounding text. The tidbit points out important information that's worth reiterating.



WARNING

This icon alerts you to a potential issue or pitfall. I point out where others have made mistakes in the hopes that it saves you time and spares some heartburn.

Beyond the Book

I can cover only so much in this short book, particularly with a complex topic. If you find yourself hungry for more knowledge on API security, just go to <https://salt.security>.

IN THIS CHAPTER

- » Taking a closer look at what APIs are
- » Recognizing how apps use APIs

Chapter **1**

Understanding APIs

If you're looking for a primer and background on APIs, you've landed on the right chapter. Here I arrive at a working definition of application programming interface (API), particularly as it relates to the internet and web design. I cover some resulting impacts of APIs to modern applications and cloud-native design.

Defining APIs

Understanding API consumer types and API protocols is key to selecting the most effective security techniques and controls available to you. *Application programming interfaces (APIs)* are sometimes used as a synonym for functions or libraries referenced regularly in code. For example, web APIs are a specific type of API designed for use in web designs and communications. Web API design patterns have existed for more than 20 years, emerging from service-oriented architecture (SOA) and back-end services powering applications via web protocols. The following sections discuss in greater detail the types of APIs and how APIs work.

Eyeing the types of APIs

The term *consumption* refers to the API caller making a request to an API to exercise functionality, query data, or manipulate data.

The range of API types has evolved substantially over the years to account for different business cases and usage models. Some of the common API types include the following:

- » **External:** These APIs support mobilized workforces and customers accessing services from anywhere. As the name implies, they're exposed to users outside a protected network or the internet, often with relaxed network restrictions. Authentication and authorization may or may not be used.
- » **Public:** Public APIs are a type of external API designed for consumption by users and machines across the internet. They have relaxed access controls or are designed for anonymous access to increase consumption.
- » **Open:** These APIs appear with open banking initiatives including the financial industry. They help promote innovation in a given industry, improve levels of service integration, and provide freedom for customers to transact or access data anywhere. Authentication and delegated authorization are usually in place.
- » **Internal:** These APIs are usually deployed and operated within a restricted network environment of a data center or private cloud segment. They're designed to be consumed by other applications or users in that network. Authentication and authorization may be in place but may be relaxed because exposure is limited.
- » **Partner:** Organizations sometimes provide limited access to internal APIs to select external suppliers to power and expand their digital supply chains. The extent of access control lies somewhere between that of internal and external APIs.
- » **Third-party:** These APIs, often consumed as cloud-delivered services, or software as a service (SaaS), help organizations move faster without re-creating functionality or incurring more technical debt.
- » **Acquired:** These are less of a design choice and more of a type of inheritance. Organizations inherit these APIs as a type of dependency as they acquire, integrate, and deploy commercial and open-source software packages.

Identifying API protocols

API protocols are the rules and specifications that govern how applications communicate with each other. They impact the design, testing, and security of your APIs. Two of the most common protocols are REST and GraphQL:

- » **REST (Representational State Transfer):** REST is a widely used architectural style that emphasizes a clear separation between clients and servers. It relies on standard HTTP methods (GET, POST, PUT, DELETE) to interact with resources. RESTful APIs are known for their flexibility and scalability, making them suitable for various applications.
- » **GraphQL:** GraphQL is a query language that allows clients to request specific data they need, reducing over-fetching and improving efficiency. It's particularly well-suited for complex applications with evolving data requirements.

While REST and GraphQL dominate the API landscape, you might also encounter older protocols like RPC (Remote Procedure Call) and SOAP (Simple Object Access Protocol). These protocols are less common in modern web development but may still exist in legacy systems.

Designing Modern Applications

APIs have significantly changed the way that development teams create applications. Whether your organization is building, acquiring, or integrating APIs, the impact to front-end design, business functionality, and data exchange is significant. APIs are also prominent within microservices architecture (MSA) and cloud-native design patterns. Continue reading the following sections for more details.

Decoupling front and back ends

Contemporary applications typically distinguish their front-end interfaces (visible to users) from their back-end services (the underlying logic). This separation enhances flexibility and maintainability. APIs are vital for linking these two elements.

Front-end code may be a script executed in a web browser or a native mobile application. Although securing front-end code is crucial, organizations possess limited authority over user devices, particularly regarding customer-facing applications.

Altering the API picture

Cloud-native has a few meanings depending on what IT circle you sit in. The broadest definition is that a design or architecture exhibits cloud traits and uses technologies that power cloud service providers. Common cloud traits include web-scale capacity and elasticity.

The following technologies enable cloud computing:

- » **Virtualization:** *Virtualization* abstracts hardware from the operating system using a hypervisor. You can run many virtual machines on a given physical host, often described in terms of density. By packing more virtual machines on a given host, you can make better use of the hardware and ensure it doesn't sit idle. Virtual machines are used readily to power applications and services, but virtual machines must be lightweight and highly performant if they're to service MSAs.
- » **Containerization:** *Containerization* involves packaging applications and their dependencies into containers to further increase density by abstracting the operating system from applications and services. Containers improve portability and environment consistency. Containers are often used as the unit of compute to power microservices within MSA.

Two most common ways that cloud-native maps to APIs are

- » Organizations pair containerization and virtualization to limit the blast radius in the event of compromised API code, container runtimes, or hypervisors.
- » Entire infrastructures can be declared and operated via APIs, for example, with container platforms like Kubernetes and cloud service providers.

IN THIS CHAPTER

- » Identifying problems with API documentation and analysis
- » Understanding API security testing methods
- » Mediating APIs to improve observability and enforce policies

Chapter 2

Laying the Foundations for API Security

Traditional approaches to securing APIs are numerous, including testing and mediation. This chapter examines these approaches in greater detail.

Documenting APIs

API documentation serves a range of security and nonsecurity purposes throughout the API life cycle. Documentation gives your organization a couple of primary advantages:

- » It provides details on how to communicate with APIs, the functionality they provide, and the data they exchange so you can better understand your API attack surface.
- » It serves as input into other activities, including design reviews, security testing, operations, and protection.



WARNING

Like all forms of documentation, teams inevitably neglect to document APIs or new functionality as they iterate. This reality leads to a type of *environment drift*, also referred to as *API drift*, that leaves massive gaps in your API inventory and security posture.



TIP

Document APIs that you build and use that data to feed an API inventory or catalog. Include third-party APIs where adequate documentation is provided by the supplier. Mitigate gaps in your API inventory by continuously scanning environments and analyzing traffic to discover new API endpoints and functions.

The following sections touch on a couple more points your organization needs to remember as it documents its APIs.

Steering clear of traditional documentation approaches

Avoid approaching API documentation as a traditional documentation exercise with the goal of producing lengthy text documents or visual diagrams as artifacts. Written documents, slides, or visual diagrams are sometimes required for compliance or as part of design reviews.

Some organizations tend to carry over security from traditional compliance and waterfall approaches. However, many IT organizations start to feel increased pressure as development teams adopt agile methodologies and DevOps practices that often come as part of a package deal with API development.

Traditional documentation can be useful for secure design reviews and threat modeling. However, traditional forms of documentation are notoriously difficult to generate and maintain. API drift can be worse in these cases, and API documentation likely won't reflect the reality of your production deployment, even with extra staffing allocated to documenting all changes.

Working with API schema definitions

The API schema definition formats are designed to make your life easier. Schema can be defined and documented during API creation. The schema definitions are also reusable for testing, integration, publishing, and operations. Many design, mocking, and development tools can autogenerate API schema definitions as you integrate or code an API.

You can use these features to reduce documentation workload and avoid headaches later. Open-source software packages that include web APIs also commonly include the relevant API schema definitions in the corresponding code repository (such as git) or package manager (such as npm) you obtain them from.

Most commonly for REST APIs, these machine formats include Swagger or OpenAPI specification (OAS). Depending on your API design, development, or publishing tooling, other formats like RAML or API Blueprint may be present. And if you're exploring GraphQL APIs, then also expect to work with GraphQL schema definitions.

Testing APIs

A specific focus within shift-left API security practices is *securing the build pipeline*, which requires that teams get security tooling plugged into continuous integration/continuous delivery (CI/CD) build pipelines and git-based developer workflows. Securing build pipelines requires a range of security-testing tools, including dependency analyzers, static analyzers, dynamic analyzers, schema validators, fuzzers, and vulnerability scanners. The type of security tooling that is needed varies based on what artifacts are moving through the pipeline, what must be built, and where it must be delivered.

The following sections examine the advantages and disadvantages of security testing within build pipelines.

Utilizing application security testing tools

Static application security testing (SAST) can be used to analyze original source code for potential weaknesses and vulnerabilities. It's often run when code is committed to version control or during build stages. Meanwhile, *dynamic application security testing (DAST)* can be used to analyze a running application for exploitable conditions. It's often initiated prior to production delivery or used in production continuously because the application must be running on infrastructure.



WARNING

SAST and DAST can uncover weaknesses and exploitable conditions in your custom API code. However, these scanning methods can't uncover business logic flaws that attackers target and abuse. Business logic — and how you design and code APIs — is unique to your organization. As a result, the code that represents your business logic rarely follows well-defined patterns where SAST or DAST signatures can be built accordingly.

Most tools don't go deep into testing authentication or authorization beyond cursory checks, such as detecting weak forms of authentication like basic and digest access or the testing tool may only analyze how credentials are input, passed, or stored. Some DAST tools can check for privilege escalation weaknesses, but doing so requires multiple runs of the tool against a given app and its API. Unfortunately, many organizations push tight release windows and time is at a premium.



TIP

DAST tools are notorious for running for extended periods for complex applications. Budget time accordingly for build pipeline scans to complete or make scans nonblocking so as not to hold up releases.

SAST and DAST have always had their shortcomings. The problem is worsened in the world of APIs. Yes, you should run them against your custom application and API code but acknowledge that these scanners aren't designed to detect all types of issues. The importance of behavior analysis in runtime for APIs can't be stressed enough (refer to Chapter 5 for more details).

Using API schema validators

A form of static analysis, API schema validators are often pitched as the DevOps-friendly solution for build pipeline security. The pitch often goes like this: "Give us your schema definitions; we can scan your APIs, make sure they're conformant, and check for vulnerabilities."



REMEMBER

However, you should be aware of a few issues that exist with the schema validation approach:

- » **Not everything needs to be defined in API schema.** API specification formats like OAS and Swagger don't require you to define all fields or functions in the API documentation. Developers commonly forget to document something fully, particularly if they aren't working within API design tools like Postman.
- » **Many organizations are lackluster at documenting.** Humans are notoriously bad at documenting, especially documenting everything fully. A lack of documentation isn't a problem specific to developers. OAS can help in that it's self-documenting, but it still requires manual effort. Some

tooling may also be better at generating the OAS definition than others.

» **API drift happens as a matter of course.** Deviations from the original specification and what is running in production are common. API drift parallels one of the biggest problems that organizations run into with secure design review and threat modeling processes. Sometimes, what you intend to build ends up looking much different than the real-world product.



REMEMBER

Any AST or schema analysis tooling you select should be integrated and automated to serve the pipeline. Tools should work within established git workflow and CI/CD processes. Scanning manually, finding issues, and spitting out a report aren't enough. Such an approach won't work for agile methodologies or DevOps practices that most organizations are embracing.

Schema validation and enforcement is the old paradigm of positive security in new clothes. Instead of security teams having to create rules or signatures, the burden is shifted to development teams. Schema validation can only identify some exploitable conditions and misconfigurations. Schema analysis can't identify business logic flaws.

Mediating 101: APIs and the Relevant Mechanisms

Although it's possible to directly expose an API via a web or application server, this practice is less common in typical enterprise architectures. API mediation can be achieved through several other mechanisms, including network load balancers, application delivery controllers, Kubernetes ingress controllers, sidecar proxies, and service mesh ingresses.

Understanding that you can mediate and observe API traffic at multiple points in an enterprise architecture is what you need to know here. Having a basic idea of the various mediation points is critical to API security so you're seeing all traffic, are able to enforce as appropriate, and remediate quickly. The actual selection of a particular mediation mechanism or the point in an architecture where you'd elect to proxy API traffic is beyond the scope of this book.

These sections examine in plain English what you need to know about mediation and APIs.

Functioning as API mediation points

In most enterprise architectures, you'll find a mix of proxying mechanisms that mediate API requests and responses:

- » **Network load balancers (NLB):** *NLBs* may be physical hardware or software-defined, and they're responsible for routing requests dynamically between servers and services to balance network load.
- » **Application delivery controllers (ADC):** *ADCs* are like their NLB counterparts, though they typically include more functionality focused on application-specific routing, load-balancing, and caching. In some cases, ADC and NLB are interchangeable and can just be a matter of vendor language.
- » **API gateways:** *API gateways* are designed specifically for mediating API traffic. They can also help with message translation and bridging between different protocols for inner and outer architecture.
- » **API management (APIM):** *APIM* suites provide full life cycle capabilities on top of API gateways alone. They typically enable an organization to unify policies across gateways and API endpoints, provide rolled-up monitoring, and enable developer or partner self-service. APIM still relies on API gateways as a mediating proxy mechanism to enable their functionality and enforce policies.



REMEMBER

If an organization is rapidly embracing microservices architecture (MSA), more than likely the organization has ingress controllers for Kubernetes and service meshes. Such ingresses are a mashup of API gateways and NLBs, and they may support multiple protocols depending on the design of the microservices. Proxies may also be deployed locally to a given workload that powers a microservice or API. Such proxies are typically referred to as *sidecar proxies*. Sidecar proxies function as a type of loopback proxy, where all traffic flowing into or out of a workload must go through the sidecar proxy.

Enforcing policy with API management

APIM offerings usually provide a set of capabilities to satisfy some use cases. Those capabilities commonly include the following:

- » **Network security:** Like NLBs or ADCs, network connectivity can be restricted to APIs in an API gateway. Commonly, this includes IP address allow and deny lists to restrict which origin IP addresses can communicate to a given API, rate limits to restrict how frequently requests can be made, and transport encryption with transport layer security (TLS) to provide confidentiality and integrity of messages in transit.
- » **Authentication and authorization:** Organizations often enforce access control at API gateways so that API calls are authenticated and authorized. Common protocols include OIDC for authentication and OAuth2 for authorization. Token translation is usually offered as well, such as where an API implementation requires integration with older protocols like security assertion markup language (SAML).
- » **Basic threat protection:** By design, the mediation mechanisms already offer message filtering and protocol translation. Many APIM offerings provide basic rules to block malicious character sets commonly used in injection attacks. The other threat protection capability includes restricting incoming API requests based on API schema definitions or manual configuration. Restrictions can be placed on parameter lengths, parameter values, array sizes, and more.



WARNING

Enable APIM security controls when possible if doing so doesn't break API integrations with other systems. These settings can mitigate some types of API attacks, but they won't protect you from most forms of API abuse and business logic attacks.

IN THIS CHAPTER

- » Identifying essential elements of API posture governance
- » Considering best practices
- » Exploring what API gateways and mediation tools can do

Chapter 3

Governing Your API Posture

API posture governance ensures your APIs are secure, reliable, and compliant throughout their lifecycle. It's about visibility into your API landscape, understanding the risks, and implementing controls to mitigate them. APIs are the backbone of modern applications, connecting systems and enabling data exchange. But with increased connectivity comes increased risk. As this chapter discusses, API posture governance is essential for protecting your organization from API-related risks.

Recognizing Key Components

API posture governance encompasses several key components, each crucial for maintaining a robust API security program:

- » **Visibility:** You need a clear picture of your API landscape. It includes knowing what APIs you have, where they are, and what they do. Visibility can be achieved through maintaining an up-to-date API inventory and employing continuous discovery mechanisms to identify new APIs as they're introduced.

- » **Risk assessment:** Understanding the risks associated with your APIs is paramount. This involves regular security testing, vulnerability scanning, and risk analysis to identify potential vulnerabilities and threats. The OWASP API Security Top 10 (refer to Chapter 4) provides a valuable framework for categorizing and understanding common API vulnerabilities.
- » **Controls and policies:** Establishing and enforcing security policies and controls is the cornerstone of API posture governance. This includes implementing strong authentication and authorization mechanisms, rate limiting to prevent abuse, and input validation to mitigate injection attacks. API gateways and other mediation tools can be crucial in enforcing these controls.
- » **Monitoring and response:** Continuous monitoring of your APIs for suspicious activity is vital for detecting and responding to attacks in real-time. An incident response plan ensures that your organization can address security incidents, minimizing damage and downtime.
- » **Compliance:** Regulatory compliance is a critical aspect of API posture governance. Ensure that your APIs comply with relevant regulations and standards, such as GDPR, HIPAA, and PCI DSS, which includes protecting sensitive data, ensuring secure storage and transmission, and adhering to industry best practices.

Identifying Best Practices

In addition to the key components, consider these best practices for effective API posture governance:

- » **Establish a dedicated API governance team.** Define clear roles and responsibilities for API security. This team should include development, operations, and security team representatives to ensure a holistic approach.
- » **Implement a comprehensive API security strategy.** Your strategy should cover the entire API lifecycle, from design and development to deployment and retirement. Ensure it addresses security testing, vulnerability management, threat protection, and incident response.
- » **Embrace automation.** Leverage automation to streamline API governance tasks, including automating API discovery, security

testing, and incident response. Automation reduces manual effort, improves efficiency, and enhances your security posture.

- » **Regularly review and update your API governance framework.** The threat landscape constantly evolves, and your API governance framework should adapt accordingly. Periodically review and update your policies, controls, and processes to address new threats and business requirements.

Noting the Role of API Gateways and Mediation Tools

API gateways and other mediation tools are essential for enforcing API posture governance policies. They act as central points of control for API traffic, enabling you to implement security measures such as:

- » **Authentication and authorization:** Verify the identity of API consumers and ensure that they have the necessary permissions to access API resources.
- » **Rate limiting:** Prevent abuse and protect your backend systems from overload by controlling the rate of API requests.
- » **Input validation:** Sanitize API inputs to mitigate injection attacks and other malicious attempts to compromise your systems.
- » **Threat protection:** Detect and block malicious traffic based on known attack patterns and suspicious behavior.

Considering Other Key Points

API posture governance isn't a one-time activity but an ongoing process that requires continuous attention and improvement. Consider the following points to ensure your API posture governance is complete:

- » **API discovery and inventory management:** Maintaining an accurate and up-to-date API inventory is fundamental to API posture governance. API discovery tools can help you identify and catalog all your APIs, including those that may

be undocumented or forgotten (referred to as *shadow APIs*). This inventory provides a foundation for risk assessment, security testing, and incident response.

» **Continuous monitoring and anomaly detection:**

Continuous monitoring of API activity is crucial for detecting suspicious behavior and potential attacks. Anomaly detection tools can help you identify deviations from normal API usage patterns, alerting you to potential threats in real-time. This approach enables you to respond quickly and minimize the impact of security incidents.

» **Misconfigurations and API posture governance:**

Misconfigurations often lead to API vulnerabilities. These can manifest as errors in authentication settings, authorization policies, or input validation rules. Effective API posture governance is vital for identifying and addressing these misconfigurations. By continuously monitoring your API environment and performing regular security assessments, you can detect and remediate misconfigurations before attackers can exploit them.

» **Data exposure and API posture governance:** Data

exposure poses a significant risk associated with APIs, as they often provide access to sensitive information, including personally identifiable information (PII) and financial records. To mitigate this risk, implementing effective API posture governance is essential.

One common vulnerability in APIs is excessive data exposure, which occurs when APIs return more data than necessary for a client application to function properly. This issue can arise from improper API configuration failing to filter data correctly or if the client application doesn't specify the required fields.

To prevent excessive data exposure, API posture governance should ensure that APIs are configured to return only the necessary data for the client application. This can be accomplished through various techniques, including data filtering, data masking, and encryption. By employing these methods, you can protect sensitive information from unauthorized access.

» **Collaboration and communication:** Effective API posture governance requires collaboration and communication across different teams within your organization. Development teams need to be aware of security best practices, while security teams need to understand the business requirements for APIs. Regular communication and knowledge sharing are essential for maintaining a robust API security program.



TIP

IN THIS CHAPTER

- » Differentiating between API attacks and application attacks
- » Looking at the OWASP API Security Top 10
- » Identifying patterns of automated attacks

Chapter 4

Getting the Lowdown on API Attacks

This chapter clarifies in plain English how API attacks differ from application attacks. You also can read more about the OWASP API Security Top 10, which is a good starting point for understanding common API flaws. Furthermore, this chapter covers automated attack patterns like brute forcing, credential stuffing, and scraping that almost always target APIs specifically.

Understanding How API Attacks Differ from Application Attacks

API attack patterns vary from what practitioners are used to within the network security and application security domains. Attacks may borrow from both domains, or more commonly, they're unique to API use cases and business logic specifically.

Attackers exploit misconfigurations in infrastructure controls, vulnerabilities in code, or some combination of the two. Leaving security to a development team was already a poor strategy, and some shift-left approaches and misguided DevOps practices have pushed responsibility too heavily onto development teams.

Developers may lack expertise in infrastructure and security concepts, which inevitably results in gaps in API security.

Attackers use your front-end applications to connect to your back-end APIs and to help decipher your business logic. Attackers also relish the fact that modern applications are highly interconnected with many first-party and third-party APIs, any one of which might be exploitable.

The front-end application is only a means to an end

Security teams sometimes attempt to secure APIs by protecting endpoints or hardening client applications. For some API consumption scenarios, you simply can't secure the endpoint or trust that client code won't be tampered with. This reality hits especially hard for deployments where customers call external APIs, public APIs, and open APIs from unsecured networks such as the Internet.



WARNING

Attackers regularly reverse-engineer client-side code or front-end code with tools that unpack, decompile, or disassemble application binaries. Attackers also make use of intercepting proxy tools such as Burp Suite or OWASP Zed Attack Proxy. These are the same proxying tools in use for general application troubleshooting and security assessment work. In the hands of a trained professional as well as an attacker, the tools are incredibly powerful. Be extra careful; always presume an endpoint is compromised along with the client-side code that runs on it. Back-end services (APIs) and the data they provide are the most valuable targets to attackers.

APIs underpin digital supply chains

Your organization's API ecosystem is more than just the APIs it builds. API integrations and API dependencies in acquired applications or online services round out any organization's portfolio. Collectively, all of these APIs form an expansive digital supply chain and increase the attack surface for organizations. The mixture of first-party and third-party APIs and infrastructure complicates what security controls are available to you, let alone what code may be visible.



WARNING

Attackers know this reality of distributed architectures and supplier integrations, and they often target the weakest link. In fact, attackers commonly exploit a weakness in an API as an initial attack vector and then pivot to other networks, servers, workloads, applications, and APIs. These multistep attack sequences often evade traditional security controls. The realities of API ecosystems further emphasize the need for runtime behavior analysis to detect novel exploits. Chapter 5 dives deeper into behavior analysis and runtime protection.

Taking a Closer Look at the OWASP API Security Top 10

Over the years, the Open Web Application Security Project (OWASP) has become increasingly popular, and its Application Security Top 10 is frequently referenced in the security industry. In 2023, OWASP updated and released the revised API Security Top 10, which identifies the ten most common flaws found in APIs. This list should be your initial reference point for gaining insights into the common weaknesses and vulnerabilities observed in APIs. Additionally, it serves as a valuable resource for training and raising awareness while also providing a lightweight framework for categorizing issues encountered in APIs.



REMEMBER

The sphere of concern over APIs shouldn't begin and end with the OWASP API Security Top 10. Attackers chain together exploits of flaws described in the OWASP API Security Top 10. They also employ automation to increase their chances of success and cause greater damage. For example, excessive data exposure and broken authentication leave an API susceptible to automated attacks such as enumeration and scraping. APIs designed in such a way are sometimes referred to as *leaky APIs*.

These sections discuss the OWASP API Security Top 10 sequentially because there's no easy way to group the flaws. Some entries like authentication, authorization, and injection flaws can seem redundant to the OWASP Application Security Top 10, but API context adds uniqueness. Other entries like improper assets management or insufficient logging and monitoring are fundamental to all systems engineering work. However, the problems become more pronounced with APIs because of rapid development cycles, undocumented API changes, and ongoing integration work.

API1:2023 Broken object level authorization

Object level authorization is an access control mechanism used to validate an API caller's ability to access a given object. Even if an application implements proper authorization checks in infrastructure, developers often forget to apply these checks before allowing access.

Attackers easily exploit API endpoints that are vulnerable to *broken object level authorization (BOLA)* by manipulating the ID of an object that is sent within an API request. These vulnerabilities are extremely common in API-based applications because the server component often doesn't fully track the client's state. Instead, the server component relies on parameters like object IDs sent from the client to decide which objects can be accessed.

Every API endpoint that receives an ID of an object and performs any type of action on the object should implement object level authorization checks. These checks should be made continuously throughout a given session to validate that the authenticated user has access to perform the requested action on a requested object.

Failure to enforce authorization at the object level can lead to data exfiltration as well as unauthorized viewing, modification, or destruction of data. BOLA can also lead to full account takeover; for example, attackers can compromise a password reset flow and reset credentials of an account for which they aren't authorized.

API2:2023 Broken authentication

Authentication mechanisms are easy targets for attackers, particularly if the authentication mechanisms are fully exposed or public. Prompting users or machines for authentication material may also not be possible in some API use cases. These two points make the authentication component potentially vulnerable to many exploits.



REMEMBER

Broken authentication in APIs originates from the following:

- » **Lack of protection mechanisms:** The API endpoint lacks an authentication mechanism. This is a common occurrence within internal networks or middleware.

» **Improper or misconfigured authentication:** The mechanism is used or implemented without considering the attack vectors, or the mechanism isn't appropriate for the use case. For example, an authentication mechanism designed for Internet of Things devices typically isn't the right choice for a web application like an ecommerce site.

This OWASP entry is a catch-all for all types of authentication issues including weak password complexity, inadequate account lockout thresholds, authentication material exposed in URLs, authentication tokens with insufficient entropy, use of API keys as the only authentication material, and lack of two-factor authentication (2FA).



WARNING

Attackers who are able to exploit vulnerabilities in authentication mechanisms successfully can take over user accounts, gain unauthorized access to data, make unauthorized transactions as other users, and abuse implicitly trusted connections to pivot attacks to other systems.

API3:2023 Broken object property level authorization

Teams sometimes design applications such that back-end APIs provide all the data that might be necessary for a given function and then depend on client-side code to filter appropriately. Because APIs can be used as a means of data exchange for many types of API consumers, back-end engineers may implement APIs in a generic way without thinking about the sensitivity or privacy of data. Traditional security scanning and runtime detection tools can't differentiate between legitimate data returned from the API and sensitive data that shouldn't be returned. This level of analysis requires a deep understanding of the application design and API context.

The weaknesses of excessive data exposure and mass assignment can be easily exploited. Here's how it happens: Attackers monitor the API traffic coming from a user interface, which can be an application on a device like a laptop or smartphone. They use the same types of tools that security experts use to intercept and view the data exchanged between the front-end (the user-facing application) and the back-end APIs (servers). After the attackers have access to this traffic, they scrutinize the API responses for data that the application receives but doesn't show to the user.

They also look for vulnerable API *endpoints* — these are points in the server that process requests — that allow a user to modify or delete sensitive data they shouldn't have access to.

API4:2023 Unrestricted resource consumption

API requests consume back-end resources such as network, CPU, memory, and storage. APIs don't always impose restrictions on the size or number of resources that can be requested by the client or user. Lack of rate and resource limiting doesn't just potentially impact performance of back-end compute though. Lack of limiting also opens the door to many types of attacks, including Denial of Service (DoS), brute-forcing, enumeration, and credential stuffing.

Here is a closer look at how attackers exploit APIs that lack limits:

- » **Lack of resource limit:** Attackers exploit lack of resource limiting by crafting a single API call that can overwhelm an application, impacting the application's performance and responsiveness or causing it to become unresponsive. This type of attack is sometimes referred to as an *application-level DoS*. Lack of resource limits may leave the system, application, or API susceptible to authentication attacks and data exfiltration attacks.
- » **Lack of rate limit:** Attackers exploit the lack of rate limiting by crafting and submitting high volumes of API requests to overwhelm system resources, brute force login credentials, quickly enumerate through large data sets, or exfiltrate large amounts of data.

API5:2023 Broken function level authorization

Broken function level authorization (BFLA) shares some similarities to BOLA, though the target with BFLA is API functions as opposed to objects that APIs interact with as in the case of BOLA. Attackers attempt to exploit both BOLA and BFLA when targeting APIs in order to escalate privileges horizontally or vertically.

Attackers discover BFLA flaws because API calls are structured and predictable. Finding vulnerable API endpoints is possible in the absence of API documentation or schema definitions by reverse engineering client-side code and intercepting application traffic. Some API endpoints might also be exposed to regular, nonprivileged users making BFLA flaws easier for attackers to discover.



WARNING

Attackers exploit BFLA flaws by sending legitimate API requests to an API endpoint that they shouldn't have access to, or by intercepting and manipulating API requests originating from client applications. For example, attackers may change an HTTP method from GET to PUT. Alternatively, attackers may also alter a query parameter or message body variable such as changing the string “users” to “admins.” Attackers exploit BFLA flaws to gain access to unauthorized resources, take over other accounts, modify accounts, or escalate privileges.

API6:2023 Unrestricted access to sensitive business flows

Unrestricted access to sensitive business flows is a type of API security vulnerability that occurs when an API doesn't properly restrict access to sensitive business flows. This weakness can allow attackers to abuse the business model behind the application to perform malicious actions, such as exfiltrating data, manipulating market or price data, or gaining an unfair advantage in competitive systems.



WARNING

Attackers typically exploit this vulnerability by automating the use of the API. For example, an attacker writes a script that repeatedly sends requests to an API endpoint to purchase all the available items of a newly released product or to scrape real-estate price information over time. This script can allow attackers to achieve their malicious intent much faster than a human user could.

An attacker can bulk purchase all items of a newly released product online and then resell them at a higher price. An attacker can scrape real-estate price information over time to predict house price trends in an area and then use this information to make profitable investments. An attacker performs actions faster than a human user on auction sites or similar competitive systems, gaining an unfair advantage.



WARNING

If this vulnerability isn't mitigated, attackers can exploit it to cause significant damage to your organization. They could exfiltrate sensitive data, such as customer records, financial information, or intellectual property, manipulate market or price data, which could lead to financial losses for your organization or its customers, or gain an unfair advantage in competitive systems, which can damage your reputation and profitability.

API7:2023 Server-side request forgery

Server-side request forgery (SSRF) is a vulnerability where an API fetches a remote resource using a URL provided by the user without proper validation. This flaw can allow an attacker to manipulate the application into sending requests to a server under their control, bypassing firewalls or VPNs.

SSRF exploits an API's feature that lets users determine which external resources it should access, such as image preview links. An attacker submits a malicious URL that causes the API to inadvertently connect to a server setup to intercept data or inject harmful content.



TIP

Attackers leveraging SSRF can steal sensitive data, overload the API server with requests, or access restricted internal resources. To prevent such attacks, organizations should validate user-supplied URLs against a whitelist, employ sandboxing to isolate API requests, and monitor API traffic for anomalies. Additionally, regular security scans, patching, and developer education on SSRF are crucial for robust API security.

API8:2023 Security misconfigurations

Security misconfigurations include insecure default configurations, incomplete configurations, open cloud storage, misconfigured HTTP headers, unnecessary HTTP methods, overly permissive cross-origin resource sharing (CORS) policies, and verbose error messages.



WARNING

Attackers exploit security misconfigurations to gain knowledge of the application and API components during reconnaissance phases, passively or stealthily gathering information about a target or victim. Detailed errors, such as stack trace errors can expose sensitive user data and system details that aid attackers in finding exploitable technology, including outdated or misconfigured

web and application servers, during their reconnaissance phase. Attackers also exploit misconfigurations to pivot their attacks, such as bypassing authentication due to misconfigured access control mechanisms.

Automated security scanners are available to detect common misconfigurations like unnecessary or legacy services. Where you detect these issues in a given technology stack varies greatly though. Vulnerability scanners may only scan a running server for known vulnerabilities and misconfigurations in published software, usually in the form of common vulnerabilities and exposures IDs (CVE IDs). However, this type of detection doesn't provide a complete picture because misconfigurations can exist in underlying code, in third-party dependencies, or in other system integrations.

Organizations often employ a barrage of security scanners in build pipelines to try to catch as many issues as possible prior to production deployment. Cases where security misconfiguration is the result of something simple like a missing patch are often minimal. Most misconfigurations that lead to exploitable APIs are far stealthier and obscured by complex architectures.

API9:2023 Improper inventory management

Maintaining a complete, accurate API inventory is critical to understanding potential exposure and risk. An outdated or incomplete inventory results in unknown gaps in the API attack surface and makes identifying older versions of APIs that should be decommissioned more difficult. Similarly, inaccurate API documentation results in risks like unknown exposure of sensitive data and makes identifying vulnerabilities that need to be remediated difficult.



REMEMBER

Unknown APIs, referred to as *shadow APIs*, and forgotten APIs, referred to as *zombie APIs*, typically aren't monitored or protected by security tools. Known API endpoints may also have unknown or undocumented functionality, which are referred to as *shadow parameters*. As a result, these APIs and the infrastructures that serve them are often unpatched and vulnerable to attacks. Attackers may gain unauthorized access to sensitive data, or even gain full server access through old, unpatched, or vulnerable versions of APIs.

API10:2023 Insufficient logging and monitoring

Insufficient logging and monitoring of APIs enables attackers to perform reconnaissance, abuse business logic, compromise systems, maintain persistence, and move laterally across environments without being detected. The longer attackers dwell or are present in an environment, the higher the likelihood the attack will result in a breach, brand or reputation damage, or some other negative impact.

Without visibility over ongoing malicious activities, attackers have ample time to perform reconnaissance, pivot to other systems, and tamper with, extract, or destroy data.

Recognizing Automated Attack Patterns

Attackers frequently create or use custom code, python scripts, command line scripts, pre-built bots, and intercepting proxies to perpetuate and automate API attacks. New attack patterns emerge as attackers abuse the unique business logic that organizations build into their APIs. These sections cover two automated attack patterns that all industries face: credential stuffing and scraping.

Brute forcing, credential stuffing, and account takeover, oh my!

Brute force attacks are where attackers enumerate through alphanumeric sequences to find working username and password combinations that provide authenticated context. Brute force attacks often combine each username in a one (username)-to-many (password) attack. Attackers may also attempt to brute force usernames, depending on how much data they're starting with.

Credential stuffing relies on lists of compromised username/password combinations and the common bad habit of users implementing the same credentials across multiple services.

Where brute forcing and credential stuffing converge is the end goal of *account takeover (ATO)*. ATO is when attackers have obtained working credentials that provide them authenticated context in a system and its APIs. Once authenticated, attackers have access to sensitive data or functionality and may try to further escalate privileges.

STOPPING CREDENTIAL STUFFING ATTACKS

Finastra, a leading FinTech platform provider, frequently defends against credential stuffing attacks, with hackers automating account ID info in an effort to succeed at account takeover. The Finastra team has deployed API security from Salt Security to automatically detect and block these attacks, which otherwise easily pass through the company's web action firewalls (WAFs) and API gateways.



TIP

Credential stuffing and brute force attacks can be mitigated by implementing policies that lock an account after multiple login attempts. However, setting aggressive lockout thresholds can impact user experience. As a compromise, organizations sometimes implement a lax lockout policy, such as locking an account after ten consecutive, failed login attempts within an hour. The failed attempt counter resets after 60 minutes.



WARNING

However, attackers take advantage of these relaxed thresholds by backing off their login requests and pausing attempts until thresholds and counters reset. This attack technique is another example of why runtime behavior analysis is necessary to detect and prevent API abuse.

The plague of scraping

Public APIs carry inherent risk because the design leans toward allowing anonymous access, and traditional access control mechanisms are a luxury. It's not possible to enforce strong authentication and authorization without registering users or employing additional authentication factors such as 2FA. Such an approach can negatively impact user experience and service adoption.

Attackers take advantage of these exposed APIs with relaxed access controls. APIs may also expose too much data or lack rate limits, which are two common flaws described in the OWASP API Security Top 10 (refer to the section "Taking a Closer Look at the OWASP API Security Top 10," earlier in this chapter). When these design flaws all apply to a given API endpoint, you end up with a leaky API that may inadvertently expose sensitive or private data. It's trivial for attackers to enumerate API endpoints and scrape data en masse using even basic scripts.

Attackers collect data at scale and in large volumes by utilizing the same data analytics tools that practitioners use to aggregate and correlate data to extract meaningful patterns. Depending on the information within the collected dataset, attackers may use scraped data to perpetuate fraud, social engineer individuals, target users with phishing attacks, or brute force accounts.

IN THIS CHAPTER

- » Acknowledging the importance of architecture
- » Continuously discovering APIs
- » Protecting APIs in runtime
- » Remediating APIs
- » Examining API security best practices

Chapter 5

Securing APIs

Traditional approaches can be beneficial to API observability and monitoring, and they also are useful for some aspects of security. However, a new approach is needed to protect APIs throughout their life cycles.

This chapter covers the importance of architecture in API security, and why a platform approach is needed to avoid the pitfalls of one-off tools and controls. I also describe the groupings of capabilities that are most critical for securing APIs throughout their life cycles: continuous discovery, protection, and remediation. This chapter also includes some API security best practices for all organizations to consider.

Recognizing Why Architecture Is Essential

The resulting gaps in API security posture that are left with traditional approaches have created the need for purpose-built API security. API security can't be addressed by a collection of splintered tools stitched together by engineering teams and operated haphazardly. This approach results in operational headaches, scaling issues, and an increased likelihood of a security incident. The following sections explain the importance of the architecture

of any API security solution and the core traits you should look for in a solution.

Focusing on architecture

Any API security tooling you consider for your organization should be built as a platform of capabilities. API security strategy demands a full life cycle approach because security issues, vulnerabilities, logic flaws, and misconfigurations arise at different stages of API design, development, delivery, and operation.



REMEMBER

API security tooling should leverage Big Data to collect and store large amounts of API telemetry, correlate API traffic, provide context, and power fast attack detection and response. The tooling should also use artificial intelligence/machine learning (AI/ML) to continuously extract useful, actionable signals for IT teams. Time-in-market is another key consideration because algorithms improve over time through training and data sets are enriched by the network effect, with more users and API calls.

Identifying core traits in an API security platform

Any API security tooling you consider should be built with automation and cloud-scale capacity in mind. Realistically, that infers a cloud-native design, making use of cloud-born technologies such as auto-scaling infrastructure components, cloud storage, and cloud analytics. This approach enables support for all your organization's environments as API adoption increases.

IN THEIR OWN WORDS

“Architecture is essential to effective API security. Only platforms with the ability to capture and analyze all your API traffic can create the context needed for full protection. You need a rich data engine and time-proven AI and ML algorithms to identify APIs and their exposed data, find and stop attackers, and distill the remediation details needed to harden vulnerable APIs.”

— Curtis Simpson, CISO, Armis

Key architecture attributes that API security tooling should exhibit include the following:

- » **Environment agnostic:** API security tooling needs to support modern and legacy infrastructures regardless of where they're hosted. The tooling should also be able to integrate with network elements like load balancers, API gateways, and web application firewalls (WAFs).
- » **Independence from additional server agents and proxies:** API security tooling shouldn't require additional server agents or network proxies. The tooling should avoid the use of client-side code to stop attacks including CAPTCHAs or JavaScript in the traffic stream. These approaches create issues with front-end performance and are ineffective in direct API communication.
- » **Cloud-based storage and analytics:** API security tooling should make use of cloud-based storage and data analytics, often referred to as Big Data. This approach is the only way to retain enough data to inform baselines of API behaviors and consumption patterns, drive analysis engines, and identify potential data loss, privacy impact, or other security incident.
- » **AI/ML-based analysis:** API security tooling should use AI/ML to analyze all the data and telemetry that are collected and produce meaningful security signals. Machine-assisted approaches are essential for powering detection and enforcement capabilities, such as determining where best to mitigate an API issue or what control is most appropriate. Machine-assisted analysis also helps reduce high false positive rates that are common in traditional approaches.



TIP

Ensure that API security tooling is designed to work in environments with encrypted transport. Some approaches suffer from reduced visibility with traffic inspection.

Incorporating data classification

APIs are used frequently to exchange data that may be sensitive. As a result, you want any API security tooling to identify any sensitive data types in API parameters and payloads as well as tag API endpoints appropriately so you're aware of potential exposures.

A range of personally identifiable information (PII) and other data types are subject to regulation. As examples, sensitive data includes protected health information (PHI) as defined by the Health Insurance Portability and Accountability Act (HIPAA) and private data includes information types defined by General Data Protection Regulation (GDPR).



WARNING

Failing to identify and protect sensitive data can result in penalties from regulatory bodies, severe brand damage, or loss of customers. API discovery can be useful for audits and prioritizing API security activities.

Protecting APIs Continuously

Chapter 2 examines how mediating technologies like API gateways can provide elements of API runtime protection. Used alone though, these mediation mechanisms leave gaps in your API security posture. Traditional runtime security approaches either aren't designed for the world of APIs, or they fail to provide full API context. You need to seek API security capabilities that can detect API attacks early and provide protection that adjusts dynamically based on your changing API attack surface.



WARNING

A common threat protection approach is to front-end API mediation points with additional proxies such as next-generation firewalls (NGFW) or WAFs. Such an approach adds latency and provides minimal to no added protection beyond the message inspection capabilities of most API gateways. Corresponding rules aren't designed for unique API business logic, and these mechanisms can't provide full context.

The following sections detail what detection and protection capabilities you should seek. Consider both sides of this coin as you select and implement any API protection.

Identifying the detection capabilities you need

You need capabilities that can identify API attacks quickly and early. WAFs and API gateways focus on transactions in isolation rather than viewing the whole picture of a complete API sequence

to provide full context. API gateways, including those that exist as components of API management and integration platforms are primarily API mediators and access control enforcers that may already be overloaded.



WARNING

Schema-dependent API security tools fail at detecting certain types of API attacks, such as broken object-level authorization (BOLA) flaws. Limitations are inherent when restricting security protection to API schema definitions at the expense of also examining traffic in runtime.

Some organizations may attempt to repurpose intrusion detection systems (IDS), intrusion prevention systems (IPS), and NGFW for API security, but these systems are even less suited for the task of API attack detection because they sacrifice any application layer or API focus for broad, multiprotocol attack detection. Detection capabilities you need include the following:

- » **Attacker correlation:** API security tooling should aggregate and correlate API traffic and associate it to attacker campaigns where applicable. Tooling should correlate attack behavior per source IP address, per user ID, and per session ID.
- » **Behavior analysis and anomaly detection:** API security tooling should programmatically parse API business logic and behaviors to assess impacts to an organization's API security posture. Tooling should exhibit traits of user and entity behavior analytics (UEBA) to detect a wide range of API abuses and automated attacks where API consumption patterns deviate from baselines.
- » **Early attacker identification:** API security tooling should continuously detect API attacks early and quickly. Attackers go through an early reconnaissance phase as they passively and stealthily probe API targets. These passive analysis techniques evade most detections because they typically appear as legitimate traffic. API security tooling should detect subtle variations in API consumption patterns that result from automation scripts and reverse engineering tools employed by attackers.

Focusing on the protection capabilities you need

Expanding on API threat protection capabilities beyond what is afforded by traditional approaches is critical. Testing or detecting issues in APIs isn't enough. You need a more comprehensive approach if you desire to stop API attacks before attackers can exfiltrate data or do damage to your organization. Here are the protection capabilities you need:

» **Stop attacks that exploit OWASP API Security Top 10 flaws.**

API security tooling should stop attackers that attempt to exploit issues defined in the OWASP API Security Top 10 (refer to Chapter 4 for more about this list). The list includes exploits of BOLA flaws, BFLA flaws, broken authentication, excessive data exposure, lack of resource or rate limiting, security misconfigurations, and injection flaws.

» **Block malicious requests while learning API logic.**

API security tooling should block or mitigate API attacks while learning the organization's unique business logic. Some API attacks can be detected and stopped regardless of how an organization designs its APIs, including injection attacks and excessive API consumption.

» **Stop credential stuffing and brute forcing attacks.**

API security tooling should stop the automated attacks (see Chapter 4) where the end goal for an attacker is account takeover (ATO). ATO is a risk for any organization that exposes an API where authentication and authorization are required. Even in cases where additional authentication factors are used, attackers combine techniques to overcome strong access controls.

» **Stop application-layer denial of service (DoS) attacks.**

API security tooling should stop application-layer DoS attacks. DoS and distributed DoS (DDoS) are often viewed from the lens of excessive traffic or request rates, or volumetric attacks. The more nefarious and stealthy form of DoS is application-layer DoS, or layer 7 DoS. Application-layer DoS is more difficult to stop because of application and API uniqueness. Ensure that the security tooling doesn't stop at layers 3 and 4 DoS. Tooling should also cover layer 7 DoS and for APIs specifically.



WARNING

Traditional rate limiting and message filtering mechanisms in API gateways or WAFs are too static, too operationally complex, or not well-maintained by the vendor. Use static limits if you have limited API consumption but seek dynamic limiting mechanisms if your API consumers are numerous or traffic patterns are less predictable.

Remediating APIs: Enhancing Capabilities and Streamlining Workflow

DevOps and DevSecOps practices reinforce the notion of feedback loops. You want your tooling to be integrated and automated in such a way that workflow is seamless. Workflow is typically centered around git-based version control systems (VCS) and continuous integration/continuous delivery (CI/CD) pipelines. Security and nonsecurity teams should be able to quickly obtain the information they need to act on and resolve issues.



REMEMBER

Remediation workflow should be minimally disruptive to normal workstreams and business activity. In practice, you can achieve this by getting as much information and resolution capability into the toolchains that are used as part of normal work.

The following sections call out remediation capabilities you need and why adapting to response workflows is necessary.

Naming remediation capabilities you need

Organizations frequently wrestle with common vulnerabilities and exposures (CVE) IDs, often generated from vulnerability scanning. However, design flaws, software weaknesses, and business logic flaws don't map neatly to CVE IDs. With respect to API security, you need to seek remediation capabilities that can check for a wide spectrum of API-related flaws, vulnerabilities, and infrastructure misconfigurations. These remediation capabilities should work continuously for the full life cycle of APIs in development, build, and runtime phases.



REMEMBER

A security fix may not always be code level, because it may not be technically possible to fix a problem in code, it may not be feasible to produce a code fix in a timely manner, or it's more practical to mitigate through other infrastructure components. Here are API remediation capabilities you need:

- » **API vulnerability and weakness identification:** API security tooling should use a combination of techniques to assess the security of all APIs in the organization. Tooling should passively analyze API traffic that flows through numerous points of enterprise architecture on- and off-premises, and it should analyze API schema definitions when available to identify areas of API weakness that should be remediated by development teams, operations teams, or both.
- » **Remediation guidance tailored to personas:** API security tooling should provide remediation guidance focused on code-level fixes for development perspectives as well as infrastructure configurations for operations perspectives. Issues should be mapped to the OWASP API Security Top 10 (refer to Chapter 4) where appropriate, but technical details shouldn't be written only for security audiences.
- » **Integration with defect tracking systems:** API security tooling should integrate with external defect tracking systems to support pre-existing security and development workflows for remediation. Defect tracking may be handled in external DevOps suites, IT services management (ITSM), or vulnerability management (VM) platforms depending on the organization's IT and security programs.
- » **Code repository and pipeline integration:** API security tooling should provide mechanisms to integrate with development, build, and release systems. Integration may be through VCS integration to statically analyze API code or schema definitions. Integration may also be through CI/CD integration to dynamically analyze APIs in runtime in preproduction or production environments.

Adapting your incident response processes for APIs

API attacks are inevitable, and your organization must deal with threat actors on multiple fronts. Even though API protection is

key to defending your APIs in runtime, your organization's ability to respond in the event of an attack is just as critical. Not all API-related risks are attack-oriented either, where dominant concerns may include data exfiltration or scraping by an attacker. Your incident response playbooks should encompass many unforeseen API events, including unintentional data exposure, privacy impacts, and availability issues.



REMEMBER

You need API-centric incident response capabilities that integrate with the work streams and tooling of development and SecOps teams. Data feeds into the organization's security information and event management (SIEM) are a given, though it should be done intelligently to provide useful signals. Integration shouldn't be limited to a basic log feed or data dump. API security tooling should intelligently prioritize events, provide actionable security alerts, and support the work streams of a modern security operations center (SOC) and IT workforce.

Defining API Security Best Practices

The broad landscape of API design patterns and API consumer types complicates security requirements for your organization. The diversity of the API landscape makes arriving at a set of best practices challenging. Your security best practices must be comprehensive and inclusive of many technology areas.

APIs are implemented, operated, or interacted with by many roles within an organization including development, API product teams, API operations teams, application security teams, and security operations. As with DevOps practices, collaboration is crucial for building and operating secure APIs. The following sections detail a range of API security best practices you should consider adopting.

API discovery and cataloging

An accurate API inventory is critical to many aspects of IT within your organization. Compliance, risk, and privacy teams require API inventory, particularly as they must answer to regulatory bodies. Security teams also need API inventory so that they can have a realistic view of their attack surface and risk posture to

help prioritize the wide range of API security activities that must be accounted for. Here are the discovery best practices:

- » Discover APIs in lower environments and not just production. Lower environments often have lax security and make for primary attack targets.
- » Include API dependencies and third-party APIs in your API catalog. Third-party APIs are part of the attack surface.
- » Tag and label APIs and microservices as a DevOps best practice. Such processes also serve as enablers of many other API life cycle activities.

Security testing

Traditional scanning technologies struggle with parsing custom-developed code and business logic because design patterns and coding practices vary per developer. Use traditional security testing tools to verify certain elements of an API implementation, such as well-known misconfigurations, vulnerabilities, and exploitable conditions, but you must operate these tools with awareness of the limitations. Testing best practices include the following:

- » Statically analyze your API code for well-defined exploitable conditions in code as it's committed to VCS, built, and delivered in CI/CD pipelines, or both.
- » Check for known vulnerable third-party dependencies and open-source componentry in your API code.
- » Dynamically analyze and fuzz deployed APIs to identify exploitable conditions in the fully integrated system.

API mediation and architecture

API mediation provides for improved visibility, accelerated delivery, increased operational flexibility, and improved enforcement capability. The latter is often used to enforce API access control. An organization can commonly achieve mediation by deploying API gateways and microgateways that function as reverse proxies, forward proxies, or both. The following are mediation best practices:

- » Mediate your APIs to improve observability and monitoring capabilities for inner and outer APIs.
- » Use mediation mechanisms like API gateways to enforce access control, rate limiting, and message filtering.
- » Augment your mediation mechanisms with API security tooling that can provide context and make the difference between static and dynamic control.

Network security

Traditional network perimeters erode as organizations move toward highly distributed APIs and cloud services. Infrastructure becomes more ephemeral, virtualized, and containerized. Consequently, this evolution makes some traditional network access control approaches ineffective. Modernized network security begins to heavily intersect with identity and access management (IAM), or “identity as the perimeter.” The following are network security best practices:

- » Enable encrypted transport to protect the data your APIs transmit over unprotected networks.
- » Use IP address allow and deny lists if you have small numbers of API consumers, such as with partner or supplier integration use cases.
- » Look to dynamic rate limiting for API deployments where API consumers are too numerous or too unpredictable.

Data security

Appropriate techniques for securing data include masking, tokenizing, or encrypting. Many data security efforts focus on securing data at rest in a back-end system, such as database encryption or field-level encryption. These encryption approaches don’t protect your data in cases where attackers obtain a credential or authorized session because the data will be decrypted for them when accessed through an API. Data security best practices include the following:

- » Use encryption selectively and as mandated by regulation due to operational complexity. Transport protection suffices for most API use cases.

- » Avoid sending too much data to API callers and relying on the API client or front-end to filter data. Sensitive or private data is always visible in traffic.
- » Adjust for modern threats like scraping or data inference where encryption isn't an effective mitigation.

Authentication and authorization

Authentication (authN) and authorization (authZ), and by extension IAM, are foundational to all security domains, including API security. IAM is used heavily for access control to functionality and data. When implementing authN and authZ, you must account for user identities as well as machine identities. Even though it's possible to challenge a user for additional authentication material in a session, this option isn't available for machine communication. AuthN and authZ best practices include the following:

- » Continuously authenticate and authorize API consumers throughout a session and based on behaviors, not just initially during login.
- » Use modern authN and authZ protocols like OpenID Connect (OIDC) and OAuth2.
- » Avoid using API keys as a sole means of authentication. API keys are primarily version control and should be paired with other authentication.

Runtime protection

Runtime protection, sometimes referred to as *threat protection*, is often delivered through proxies like API gateways and WAFs. These mechanisms rely on message filters and static signatures, which can catch some types of attacks that follow well-defined patterns but miss most forms of API abuse. Runtime protections are useful for identifying misconfigurations in API infrastructure as well as behavior anomalies like credential stuffing, brute forcing, or scraping attempts by attackers. Runtime protection best practices include the following:

- » Enable threat protection features within your API gateways and API management to mitigate risk of injection flaws like JSON and XML injection.
- » Ensure that DoS and DDoS mitigation are part of your API protection approach. If attackers can't exploit or abuse an API, they often revert to DoS techniques.
- » Augment traditional runtime controls with AI/ML and behavior analysis engines to detect novel API attacks where pre-built signatures leave gaps.

IN THIS CHAPTER

- » Designating security leads and forming a strategy
- » Protecting APIs and adapting incident response
- » Starting and maintaining an API inventory

Chapter 6

Ten Things You Can Do Now to Secure APIs

Not sure about what you can do to secure APIs in your organization? What follows is a list of ten high-priority items you should focus on to identify and protect your APIs:

- » **Identify API security leads.** Start with your application security team and identify API security leads to collaborate on discovery, testing, protection, and incident response. Your API security expertise may be scattered across development, infrastructure, operations, and security roles. You may also find that expertise is concentrated within API product teams.
- » **Develop a comprehensive API security strategy.** Treat APIs like any other critical IT asset and build a plan that addresses the entire API lifecycle, including design, development, deployment, and management. Define specific security standards for APIs to meet regulatory requirements and industry best practices. Ensure your API security strategy aligns with relevant regulations like GDPR, HIPAA, and PCI DSS.
- » **Establish an API inventory.** Build an initial API inventory and plan to maintain it as your API landscape evolves. Your API inventory is more than in asset management databases

or API management. You need discovery mechanisms to identify API endpoints, gather metadata, and classify potentially exposed data types.

- » **Prioritize and classify APIs.** To prioritize security efforts and allocate resources effectively, categorize APIs based on their risk level and sensitivity. Consider factors such as data sensitivity, business criticality, and the potential impact of a breach.
- » **Implement secure development practices and integrate security into the API development lifecycle.** This includes secure coding practices, code reviews, and CI/CD pipeline security testing.
- » **Deploy purpose-built, AI-enabled tools for runtime API protection.** These tools should provide advanced threat detection capabilities, such as anomaly detection and behavioral analysis. Integrate real-time threat intelligence feeds into your runtime protection mechanisms. Ensure comprehensive protection across all your APIs, including those deployed on-premises, in the cloud, or hybrid environments.
- » **Adapt incident response for APIs.** Augment your digital forensics and incident response processes for the world of APIs. API abuse and data exposures may not rank high on the list initially, but successful API attacks have a massive impact on organizations. Ensure that SecOps teams have what they need to respond quickly and loop in appropriate API expertise.
- » **Define the API remediation process.** Formalize your remediation steps to help power feedback loops critical in DevOps. Remediation often requires a mix of roles and can also include third parties.
- » **Embrace automation to streamline API security tasks.** Automate security testing, vulnerability scanning, and incident response.
- » **Promote collaboration and communication.** Foster collaboration between development, operations, and security teams. Encourage communication and knowledge sharing to ensure everyone knows API security risks and best practices.



Securing the API Explosion with Salt's API Protection Platform.

APIs are booming and the Salt Security platform offers complete protection.

Backed by the largest data lake in the API world, Salt finds all your APIs, makes sure they are in compliance with your policy and stops them from being attacked.

The time for Salt is now. Learn more at salt.security



Fearless API Security.

Find out what you need to secure your APIs

In today's world, organizations often think application security and security for their application program interfaces (APIs) are the same. That view is incorrect. *API Security For Dummies*, Salt Security Special Edition, examines what your organization needs to know about the differences and shares key insights on how to protect your APIs. This handy guide explains what APIs are, what security measures are unique to them, how you can look out for attacks, and what you can do to safeguard your APIs to protect customer information and other important data.

Inside...

- Recognize the different types of APIs
- Grasp the OWASP API Security Top 10
- Identify automated attacks that target APIs
- Define your API remediation process
- Understand why architecture is key
- Adopt some API security best practices



Eric Schwake has researched and advised on a range of application security and infrastructure security topics including API security, secure software development lifecycles, container security, and Kubernetes security. He has more than 20 years of hands-on practitioner and leadership experience in application security, enterprise architecture, and systems engineering.

Go to **Dummies.com™**
for videos, step-by-step photos,
how-to articles, or to shop!

ISBN: 978-1-394-33840-5

Not For Resale



9 781394 338405

WILEY END USER LICENSE AGREEMENT

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