42crunch

API **SECURITY**

A Blueprint for Success

How to build out a successful API Security program for your enterprise. Leverage your existing infrastructure, development and security teams for success.

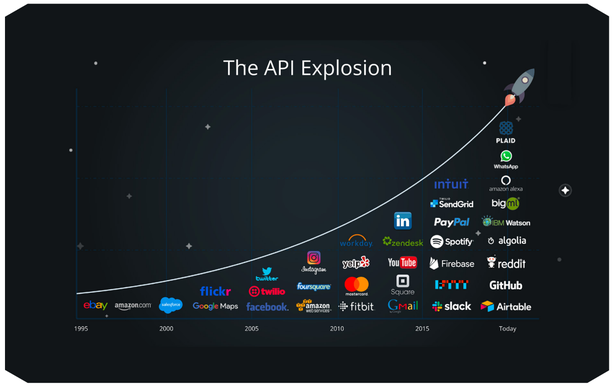
INTRODUCTION

APIs are a core building block of every enterprise's connectivity and digital strategy, yet they are also the number one attack surface for hackers. The explosion in the volume of APIs over the past ten years[1] has put such strains on traditional security and API management approaches that they are failing everyday as the scale and reach of API attacks increases.

The industry's leading API security community newsletter, apisecurity.io, has been curating these attacks on a weekly basis since 2018. It documents the ever increasing number and losses enterprises accruing from these attacks.

A recent Forbes article points out that it's time to take the API security threat more seriously because, "...ops and security teams lack a systematic way to spot, prevent and respond to API-based attacks"[2].

In this ebook we set out how enterprises can adopt a systematic approach to implementing API security **at scale**. It is a blueprint for how your organization should adopt a true DevSecOps Security program by implementing the right processes and tools for developers, operations and security teams to ensure a scalable, long lasting and automated protection of your APIs today and tomorrow.



APIsecurity.io

Powered by 42crush

THE LATEST API SECURITY NEWS, VULNERABILITIES AND BEST PRACTICES

Issue: #158

Data of 400 000 students exposed, 1 million sites affected by plugin vulnerabilities, views on GraphQL

Issue: #164

Log4Shell vulnerability, API sprawl an increasing threat, API security design best practices, Zero Trust for APIs

December 15, 2021

BY 2025

less than 50% of enterprise APIs will be managed, as explosive growth in APIs surpasses the capabilities of API management tools.

APPLICATION SECURITY TEAMS

will deploy perimeter controls with threat inspection capabilities, but will be limited to generic policies and detection signatures.[3]

1. The API Explosion. 42Crunch and Postman Webinar

2. It's Time To Take The API Security Threat More Seriously. Nov 11, 2021 https://www.forbes.com/sites/forbestechcouncil/2021/11/11/its-time-to-take-the-api-security-threat-more-seriously/

3. Predicts 2022: APIs Demand Improved Security and Management. Published 6 December 2021 - ID G00758711 By Shameen Pillai, Jeremy D'Hoinne, John Santoro, Mark O'Neill, Sham Gill

API SECURITY BLUEPRINT

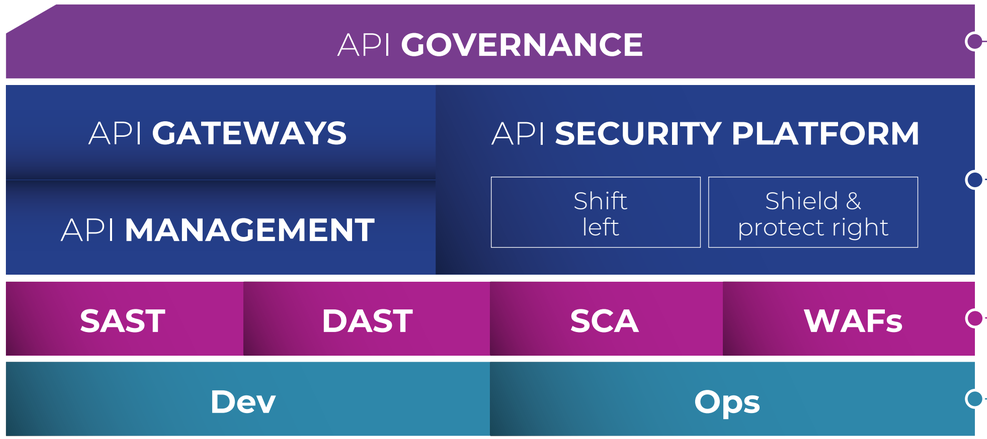
TABLE OF CONTENT

API SECURITY **BUILT UPON A SOLID FOUNDATION**

A successful API security initiative is ideally built upon a **solid foundation** of a **sound** DevOps practice and a **balanced** AppSec program.

Just like a house, the strength of the overall structure is dependent on a **solid foundation** — without these in place, an API security initiative may prove challenging.

Similarly, it is imperative that dedicated API security solutions are deployed **in addition** to AppSec tools to provide the optimum coverage and protection specific to APIs.



API management portals and gateways are essential to the operation of APIs at scale, and **security features should be leveraged to provide another layer of protection.**

A dedicated API Security platform is essential as the ultimate layer of defense allowing for:

- **Validation** of OAS specifications
- **Verification** of API implementation
- **Runtime protection** of APIs
- **API discovery** and inventory
- **Integration** into IDEs and CI/CD

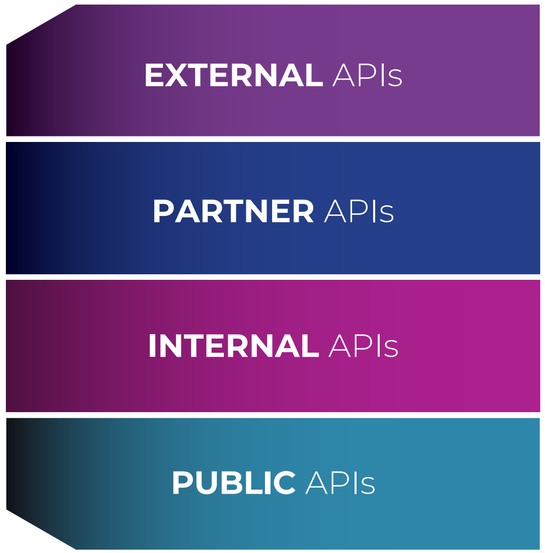
A well established AppSec process is **vital** to ensure that basic coding and implementation errors are detected in the build process.

Traditional security tools (WAF, SAST, DAST) are not specifically tailored for API development and are likely to lead to **gaps in coverage**.

DevOps emphasizes collaboration between Development and Operations teams to ensure **high-velocity delivery of quality applications in a repeatable, automated manner.**

UNDERSTANDING YOUR **API INVENTORY**

THE DIFFERENT TYPES OF API



These are **authenticated public-facing APIs** accessible via the internet. Typically, these APIs serve customers and consumers, or mobile and web applications. These APIs are critical since they are public-facing and expose critical, sensitive data. **Strong authentication and authorization are key** for these APIs.

These are APIs to **enable specific partner functionality** and have more restricted access than external APIs. For partner APIs pay attention to **data privacy requirements**.

These are APIs accessible only via **private, internal networks**. Typically, these APIs enable interoperation between various internal business applications. **Security is still key for these APIs** — assume that in time all internal APIs will become external. Do not rely on traditional perimeter protections.

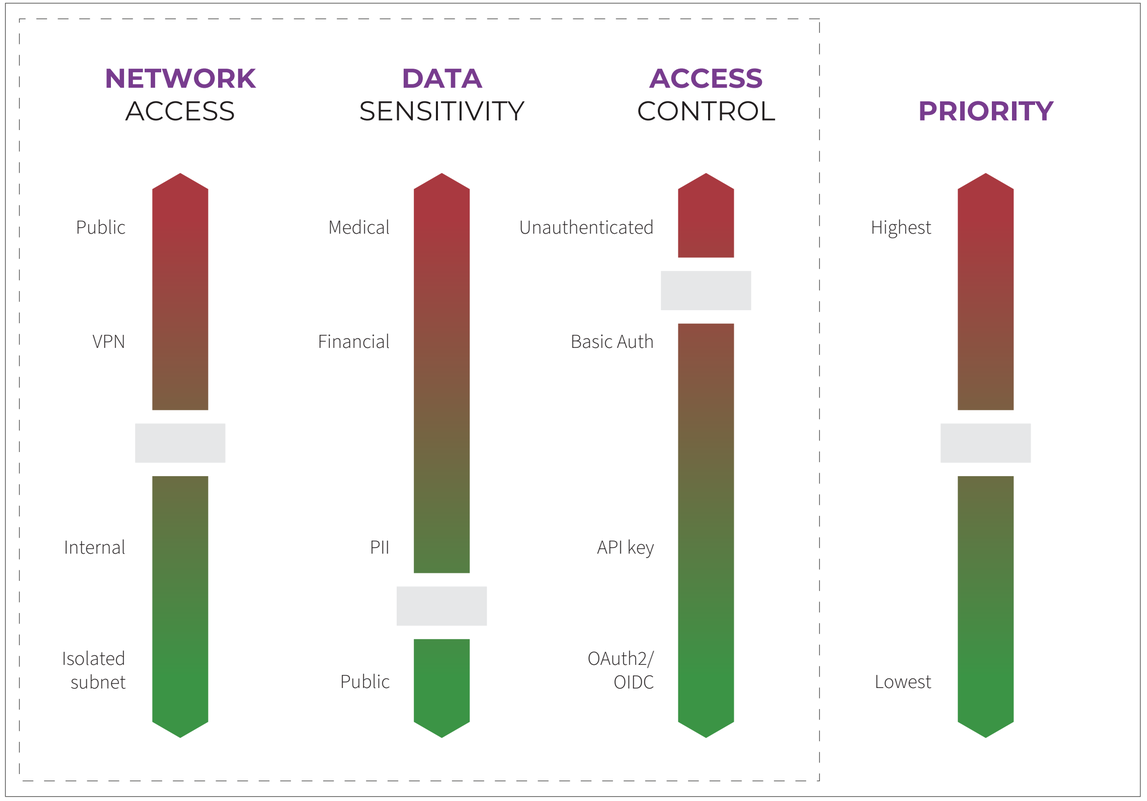
These are **open access public-facing APIs** accessible via the internet. Typically, these APIs serve unauthenticated information or status information for public consumption. For these **APIs focus on availability considerations** such as rate limiting and usage quotas.

PRIORITIZING YOUR **API INVENTORY**

Starting your API security journey **requires a focused effort on your entire API inventory**, starting with your most critical APIs. To determine the priority, perform the following three assessments:

1. Assess the **network access** to the API; public APIs are at the highest risk, and isolated networks the lowest risk.
2. Assess the **data sensitivity or classification**; medical or patient data is at the highest risk, and information in the public domain is at the lowest risk.
3. Assess the **access control** to the API; unauthenticated access is the highest risk, and strongly authenticated via OAuth2 or similar is the lowest risk.

Once a rating has been determined for each of the three categories this can be combined to provide an **approximate overall risk rating** which determines **priority**. Start at the top, and work through the inventory.



WHO IS ATTACKING YOUR APIS?

”SCRIPT KIDDIES”

Generally lower skilled attackers utilizing publicly available tools to attack APIs either for mischief or notoriety.

SCRAPERS AND BOTS

Scrapers can exfiltrate data via APIs for reselling, and bot farms can launch sophisticated large-scale attacks against public APIs.

HACKERS

This is the most dangerous attacker – highly skilled with advanced techniques. They are usually incentivized for financial gain or political/social motives.

THE DANGERS IN YOUR API INVENTORY

SHADOW APIs

These APIs are **invisible to the security team** — usually built in a clandestine manner to meet urgent business requirements. Public cloud adoption has driven shadow IT and represents an unquantified risk to an organization.

ZOMBIE APIs

This API is typically a **deprecated or outdated API** that remains active to support legacy systems. Often these APIs are not maintained or patched representing a significant risk to an organization.

MISCONFIGURED APIs

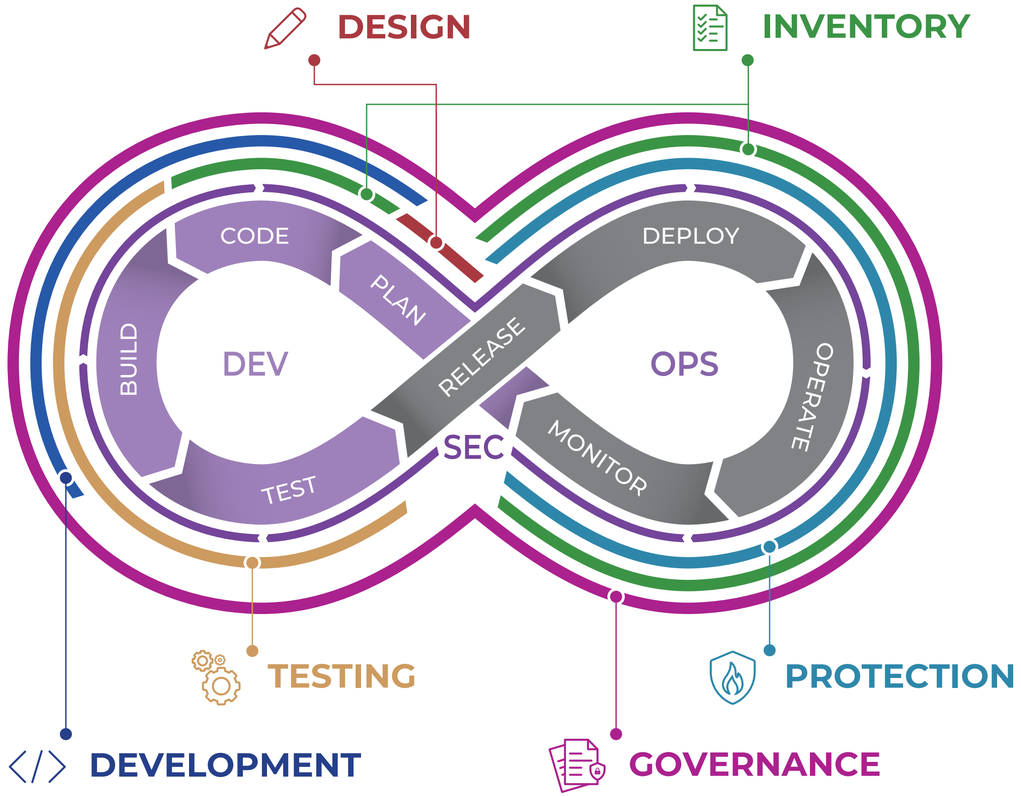
Cloud infrastructure and frameworks have fueled API growth, however, the complexities of these environments often result in **APIs that are misconfigured** (insecure defaults, missing security controls, etc.).

“FRANKENSTEIN” APIs

Similar to shadow APIs, these are **developed in a non-standard fashion** often outside of standard governance and security processes resulting in increased risk.

WHO?	HOW IT HAPPENED?	WHAT WAS THE IMPACT?
GLOBAL SHIPPING COMPANY	Researchers discovered they could automatically submit parcel numbers to an API that retrieved a map image. They then used this image to guess the postcode and then were able to retrieve full parcel information and extended user information. The underlying causes were the **lack of rate-limiting** on the parcel number API, and **excessive information exposure leading** to leakage of customer PII.	Potentially **large-scale exfiltration of customer PII and parcel tracking information**. Researchers reported responsibly and a fix was released before exploitation. The vulnerability was disclosed after all systems were patched.
CRYPTOCURRENCY TRADING PLATFORM	A researcher discovered an issue in a cryptocurrency trading platform whereby he could trade between two different accounts. The platforms failed to validate the account details and allowed purchases from accounts with insufficient funds. The exploit could be triggered by manipulating API request parameters. The underlying cause was a text-book case of **broken-object level authorization** allowing manipulation via an API parameter.	In this case, the impact was fortunately limited due to responsible disclosure and immediate response. The researcher received a **substantial bounty of $250,000.**
INSURANCE COMPANY	A researcher discovered an assortment of API vulnerabilities can be chained together to totally compromise an affected platform. This vulnerability is a great example of how multiple weaknesses can be chained together to totally compromise a system. In this case, the following issues were discovered: **UUID leakage, broken object-level authorization, broken function-level authorization, poor JWT validation, secrets committed to source code repositories, and leakage of API tokens via web UI.**	In this case, the security researcher worked with the company to verify the vulnerability and worked together on the remediation within a very short timescale. No breach of data was reported, and a small bounty was paid to the researcher.

DevSecOps **CYCLE**



API **INVENTORY**

Do you understand what APIs you own? Do you track shadow and zombie APIs?

API **TESTING**

Are you doing automated API testing? Are you considering security in your test strategy?

API **DESIGN**

Are you doing API-design-first? Do you incorporate security into the design phase?

API **PROTECTION**

Are you using API protection technology (WAFs, WAAPs, API gateways) in your deployments?

API **DEVELOPMENT**

Are your developers trained to code securely? Do they understand API security threats and risks?

API **GOVERNANCE**

Do you control and actively monitor your API estate and environments?

CORE DOMAINS OF API SECURITY

API DESIGN

WHAT IS IT?

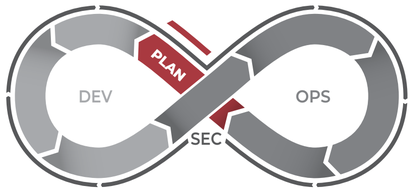
A solid API design practice is the foundation of usable, scalable, documented and secure APIs. Key elements of secure API design include:

- **Authentication** methods
- **Authorization** models and access control
- **Data privacy** requirements
- **Compliance** requirements
- Account reset mechanisms
- Use and abuse cases
- Key and token issue and revocation methods
- **Rate limiting** and **quota enforcement**

Additionally, API design teams should perform **threat modelling exercises** to understand their threat environment and attack surface.

WHY IT MATTERS?

It is significantly more cost effective to address security issues at the design phase, rather than later in the lifecycle — a **shift-left approach** is key.



API DEVELOPMENT

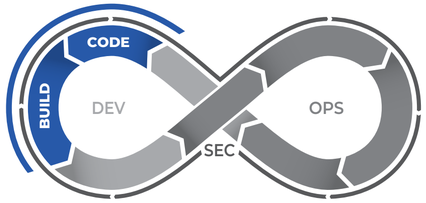
WHAT IS IT?

A crucial element of secure APIs is the development process where specifications are implemented in live APIs. Key considerations here are:

- Choice of languages, libraries, and frameworks
- Correct **configuration** of frameworks to ensure security best practice is followed
- **Defensive coding** — do not trust user input, handle all unexpected failures
- Use **central points of enforcement** of authentication and authorization – avoid 'spaghetti code'
- **Think like an attacker!**

WHY IT MATTERS?

This vital stage is where the rubber meets the road — developers should ensure they are following **security best practice** to avoid introducing vulnerabilities into APIs.



API TESTING

WHAT IS IT?

API security testing is vital to ensure that APIs are verified as secure before deployment. Security testing should be tightly integrated into the CI/CD process and should avoid any manual effort. Tests should be able to 'break the build' in event of failure.

The following aspects should be tested:

- **Authentication and authorization** bypass
- **Excessive data or information exposure**
- Handling invalid request data correctly
- Verifying response codes for success and failures
- Implementation of **rate-limiting and quotas**

WHY IT MATTERS?

Without adequate API security testing an organization runs the risk of deploying insecure APIs — **test early, test often, test everywhere.**

CORE DOMAINS OF API SECURITY

API INVENTORY

WHAT IS IT?

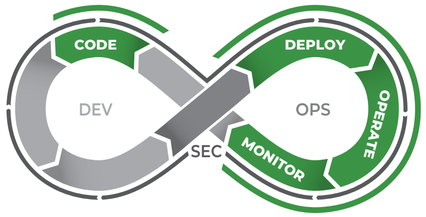
The old adage of **'you can't protect what you can't see'** applies perfectly to API security. As APIs grow exponentially fueled by business demand it is increasingly difficult for the security teams to maintain visibility of what APIs exist and what risk they expose.

Three elements are key:

- How are new APIs **introduced and tracked** in the organization?
- **Discovery** of the API inventory by introspection of source code repositories to discover hidden API artifacts
- **Runtime discovery** of APIs (via network traffic inspection, etc.) aids understanding of shadow and zombie APIs

WHY IT MATTERS?

An **up-to-date and accurate inventory** is key to maintaining visibility into the exposed risk and attack surface.



API PROTECTION

WHAT IS IT?

Despite the best efforts during the preceding phases of the SDLC, APIs will still come under attack and should be protected via dedicated API protection mechanisms.

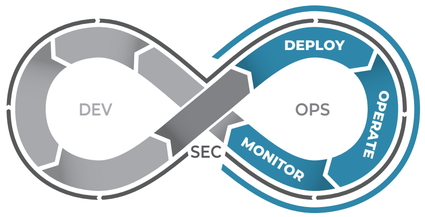
API protection should include:

- JWT validation
- Secure transport options
- Brute force protection
- Invalid path or operation access
- Rejection of invalid request data
- Filtering of response data

Protection logs should be ingested into standard SIEM/SOC platforms to ensure visibility of API security operations.

WHY IT MATTERS?

A **defense-in-depth** approach is the foundation of risk reduction — regardless of how well designed your APIs are, they will still be attacked by persistent and skilled adversaries.



API GOVERNANCE

WHAT IS IT?

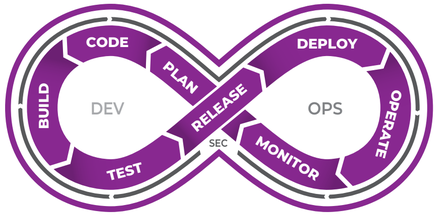
The final domain of API security is the overall governance process which ensures that APIs are designed, developed, tested, and protected according to the organization's process.

Governance covers the following principles:

- APIs are **consistent** i.e., use standard patterns for authentication and authorization
- **Standard processes** are followed for the development of new APIs
- **Data privacy and compliance requirements** are being met
- A **process** is observed for APIs at their end-of-life to eliminate insecure zombie APIs

WHY IT MATTERS?

Trust but verify — a robust governance process is essential to ensure that API development observes organizational methodologies.



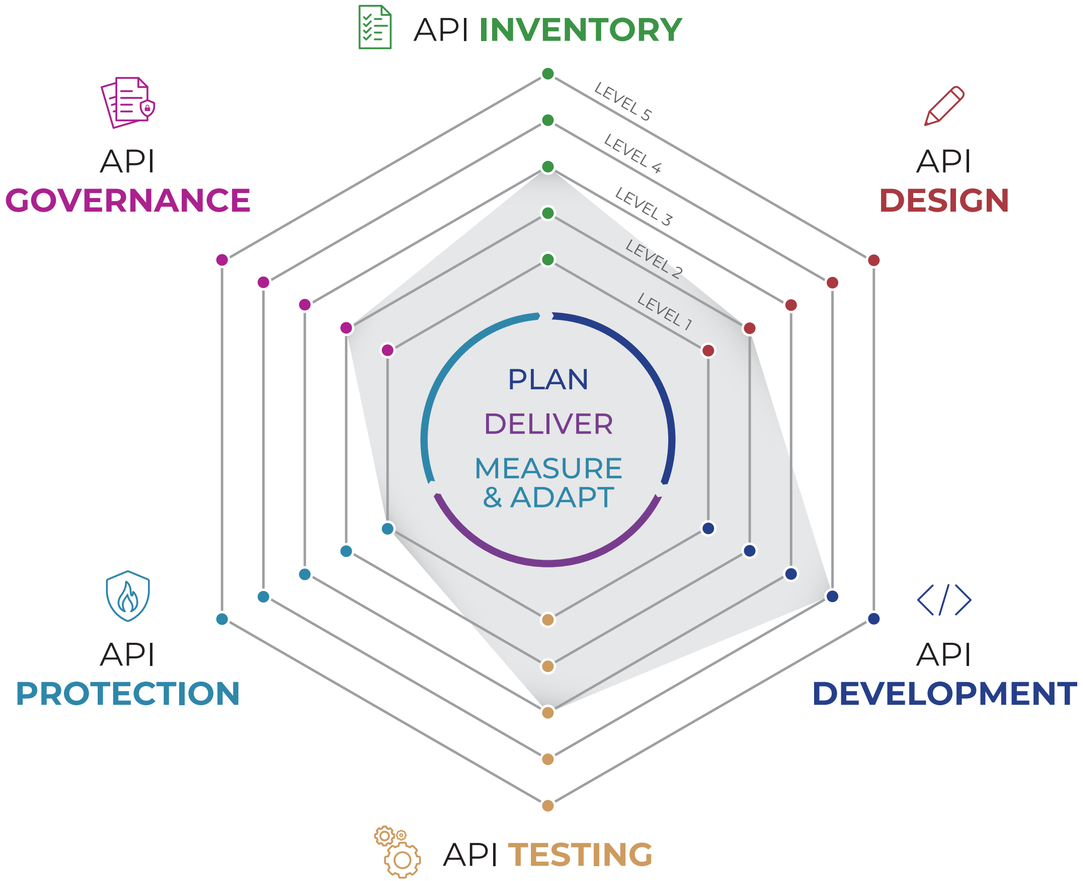
INCREASING API SECURITY MATURITY

API INVENTORY	API DESIGN	API DEVELOPMENT	API TESTING	API PROTECTION	API GOVERNANCE
Basic inventory maintained via spreadsheet or manual tracking. No management of shadow/zombie APIs.	No formal API design process in place, instead a code-first method is used. No upfront consideration of security concerns, threats, compliance, data privacy.	Developers are largely unaware of security concerns in API development or approaches to secure code in general.	No specific API security testing is performed, with only functional testing in place.	No specific API runtime protection is implemented, the only protection in place is standard firewalls or WAFs.	API development is largely ungoverned with business units each using their own process with no central oversight.
Inventory maintained via API Management or centralized platform. Standard process for new API development.	APIs are developed using a design-first approach based on OAS definitions. Security concerns are addressed on an ad-hoc basis with no standard process.	Developers have a familiarity with security considerations and use secure coding practices albeit sporadically.	API security testing is performed using largely manual testing, and lacks automation and CI/CD integration.	Some protection is provided typically using API gateways to provide basic enforcement of rate-limiting, token validation, etc.	Governance addresses only the basic requirements of compliance and regulatory requirements.
Inventory actively tracked via centralized platform, shadow and zombie APIs deprecated, and upgraded.	Security a first-class element of API design including standard patterns/practices, including threat modeling.	Developers are fully versed in secure code and API security topics, and proactively seek to use best-practice, and defensive coding.	API security testing is tightly integrated into all stages of the SDLC, and failures can block releases.	Dedicated API firewalls are implemented to provide localized protection at the API transaction level.	Governance is proactive and APIs are developed to a standard process. Deviations are tracked, and discrepancies addressed.

UNDERSTANDING YOUR **API SECURITY MATURITY**

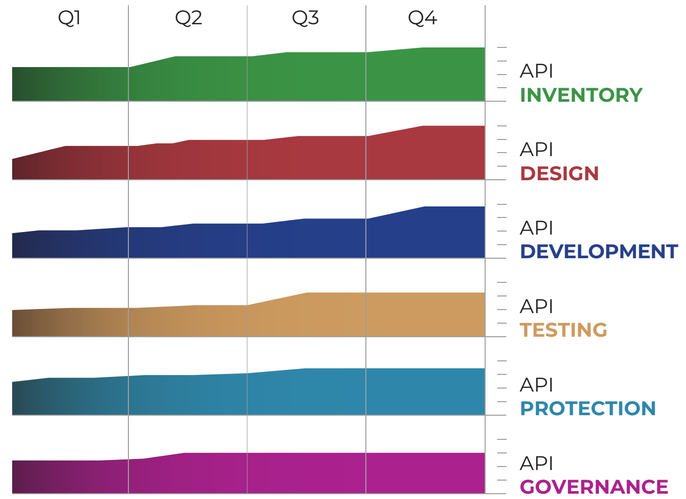
STEP 1

An assessment of current API security posture across the six domains gives a 'stake in the ground' view of current maturity.



STEP 2

Using the assessment, the organization can map (and track) a roadmap for continuous API security capability improvement.



AUTHENTICATION VULNERABILITIES

Broken or missing authentication is a leading cause of API vulnerability. A wide range of authentication mechanisms exist, and often, developers are overwhelmed when choosing and implementing a method leading to exploits in production.

Common scenarios include:

- Total unprotected endpoints.
- Leakage of tokens and keys (OAuth addresses these issues).
- Exposure of user credentials.
- Abuse of password reset mechanisms.

For every API endpoint, be sure to enforce authentication ideally via built-in middleware. If an API is public, then explicitly mark it as such in the code.

OBJECT-LEVEL VULNERABILITIES

The top vulnerability by incidence and severity is broken object-level authorization. In this vulnerability, an endpoint allows a given user access to an object (namely data) that does not belong to that user. The root cause of this vulnerability is a failure in the API back end to validate that the requesting client has access to the specified object. Typically, attackers will gain authentication to an API and then attempt to manipulate object identifiers to probe for poor implementations.

Remember: Always fully validate the permissions of a client to access a given object.

FUNCTION-LEVEL VULNERABILITIES

Broken function-level authorization occurs when a user can successfully access an API endpoint (method or function) to which they are not authorized. For example, an API might include an /admin endpoint intended for administrator-level privilege users. If a user without administrator privileges could access this endpoint, this would constitute broken function-level authorization. **The root cause is a lack of user authorization on a given endpoint.**

If fine-grained function-level access is required, ensure that the correct authorization checks are made in the relevant back-end code.

DATA VULNERABILITIES

There are two important classes of vulnerabilities associated with data — excessive data exposure and mass assignment. Both are extremely common and carry a significant risk impact on the API and underlying data.

Excessive data exposure occurs when an API returns more information than expected or necessary to meet its goal. For example, if a client attempts to retrieve user details, a poorly implemented API method might return sensitive user information such as PII data. The API should return the bare minimum of data to meet the design requirement and no more.

Mass assignment occurs when an attacker can inject additional request data, which is incorrectly interpreted by the API endpoint and stored in the underlying data record. A typical example is when an attacker guesses a field name (such as isAdmin) and assigns a value. Ideally, the back end would ignore such extraneous fields, but a vulnerability would interpret this value and store it unintentionally. In this way, an attacker can modify the API's underlying data store.

Data vulnerabilities can be easily tested and detected using conformance scanning of APIs against their definition. Developers should be aware of using the full data object and exposing this on the API — rather, a simplified data object (with sensitive fields removed) should be used for API operations.

CONFIGURATION VULNERABILITIES

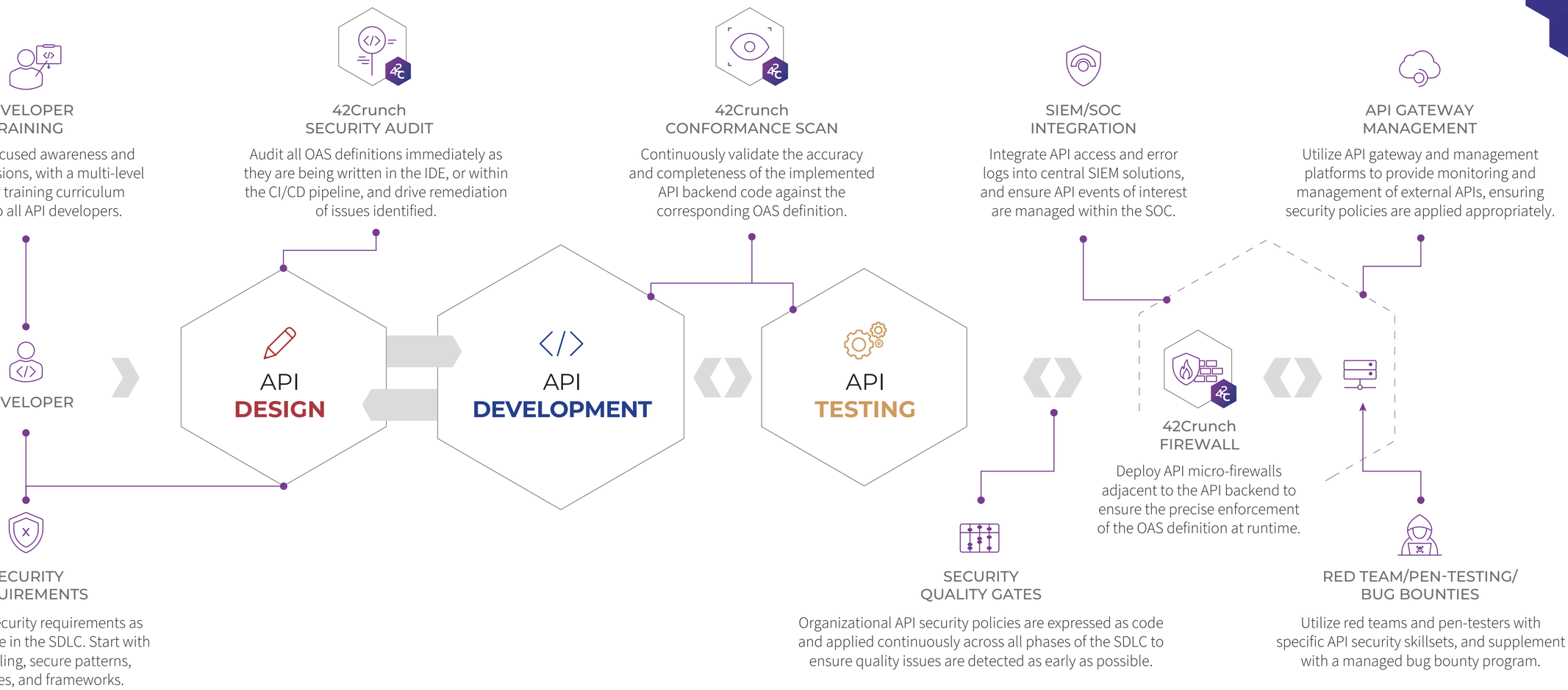
Security misconfigurations cover a wide range of operational or runtime misconfigurations, which result in a vulnerability infrastructure for an API. These can include issues relating to misconfigured headers, missing transit encryption, accepting unused HTTP methods, overly verbose error messages, and lack of sanitization.

Improper asset management occurs when an organization exercises inadequate governance over their API development process, leading to the existence of shadow or zombie APIs.

IMPLEMENTATION VULNERABILITIES

Finally, this catch-all category of vulnerabilities covers various implementation and coding vulnerabilities, including issues such as input injection (SQL, command, LDAP, etc.), lack of data sanitization, use of vulnerable libraries or components, misconfiguration of frameworks, etc.

KEY COMPONENTS OF THE SECURE API SDLC



QUICK WINS FOR IMPROVING API SECURITY

ALWAYS USE TLS

Transport Layer Security should no longer be considered a "nice-to-have" and should be enforced as a standard for all API endpoints. Care should be taken to correctly configure security headers and transport security, thus ensuring that a certificate rotation process is in place.

USE OAUTH2 FOR SSO

Poorly implemented authentication and authorization rank highest on the OWASP API Security Top 10. Developers should resist the temptation to invent their workflows for the issuance and revocation of tokens. This is so that they can leverage the collective wisdom behind the OAuth 2.0 standard. Furthermore, developers are advised to seek a deeper understanding of the various authentication flows by using an OAuth playground and draw on advice from authoritative guides to implement best practices.

PRACTICE GOOD HYGIENE WITH API KEYS AND TOKENS

Once a key or token has been obtained, developers should ensure they are practicing good hygiene, including the following:

- Avoid committing keys and tokens to version control repositories.
- Avoid storing keys and tokens in clear text either on endpoint devices or on CI/CD pipelines.
- Use short-lived keys and tokens where possible.
- Use keys and tokens with the minimum level of access required for the role. Avoid granting overly permissive privileges.

USE A POLICY AGENT TO MANAGE FINE-GRAINED AUTHORIZATION

Function-level authorization requires access controls to be implemented in the endpoint code for each function. Implementing such functionality with homegrown libraries or reusing code is fraught with danger — code can become unmanageable, and bugs in logic can easily be introduced. Best practices prescribe the use of a central authorization endpoint — to centralize decision making — or the use of an authorization framework.

ELIMINATE UNUSED METHODS ON ENDPOINTS

A quick win in reducing the attack surface on an API is to remove unused methods (i.e., HTTP methods typically such as DELETE, OPTIONS, PATCH, etc.). Depending on the framework used, invalid methods may be invoked on an endpoint causing unexpected behavior. The recommendation is to explicitly eliminate such methods using route mappings in the framework.

APPLY QUOTAS AND RATE LIMITING

The abuse of APIs can result in denial-of-service attacks or allow large-scale data exfiltration. Endpoints should use both quotas to limit excessive use or rate limiting to reduce abuse cases. Such defenses can be implemented in code, but ideally, they should also be implemented at a firewall or gateway level.

USE API FIREWALLS AND GATEWAYS

Low-hanging fruit for API security — all cloud platforms offer API gateway services, and these should be enabled with strong baseline policies applied, e.g., rate limiting and transport security. To address specific API security concerns like data exfiltration, JWT validation, etc., bespoke API micro-firewalls should be utilized.

MANAGE YOUR API INVENTORY

API attack surfaces can be dramatically reduced by active management of the organization's API inventory. Firstly, a governance process should prescribe standards and best practices for new APIs, using guidance from this article and other industry standards. Secondly, actively taking steps to identify the organization's API landscape enables an analysis of out-of-date or deprecated APIs — and APIs that are outside of the governance process.

USE PROVEN LIBRARIES FOR STANDARD FUNCTIONS

Many of the most challenging aspects of API development pertain to the implementation of common patterns or workflows — think of OAuth2 authorization code flows, JWT token validation, or fine-grained authorization policy. Although tempting for developers, they should resist the temptation to build their implementations for such patterns and rather rely on the wisdom of the crowd. Adopt a standard library, and actively participate in the community to contribute patches and fixes as you find them.

SEE RESULTS **IMMEDIATELY**

Get your development and security teams going immediately with our self-service, easy to use programs. See benefits in seconds.

SELF-SERVICE AUDIT

A free API Security Audit. Developers submit an OAS file and get an instant rating score of the vulnerabilities and formatting of API definitions.

AUDIT & DISCOVERY

Find and inspect your OpenAPI files. Connect directly to your Github repository and detect and score vulnerabilities in your API.

SANDBOX

Developers & Security teams work simultaneously without dependencies in a dedicated environment. Set pre-defined capabilities and testing scenarios.

ASSESS & MAP

Get a free assessment of where you are at today on your API Security program. Build out a roadmap to API security success.

CONTACT US TO START GENERATING VALUE RIGHT AWAY.

ABOUT **42CRUNCH**

With 42Crunch you never let unsecure APIs reach production.

We automate API security to provide continuous protection for your digital business at scale. Our unique developer-first API security platform enables developers build and automate security from inside the API development pipeline and gives security teams full visibility and control of security policy enforcement throughout the API lifecycle. Deployed by Global 2500 enterprises and used by over 500,000 developers, 42Crunch delivers security as code to enable a continuous and seamless DevSecOps experience.

JOIN THE **API SECURITY COMMUNITY**

Check out the popular APISecurity.io community website for all things related to API security. Our weekly API Security newsletter covers the latest breaches, vulnerabilities, standards, best practices, regulations, and technology.

Global office network in San Francisco - Dublin - London - Montpellier