
crunch
State of API Security
2026
Real-World Vulnerabilities, Emerging Risks, and the Future of API Security in an AI-Driven World

Introduction

The State of API Security 2026 report delivers a data-driven analysis of real-world API vulnerabilities, showing how common mistakes in implementation translate into security risks in production. Drawing on extensive cases from a wide range of independent sources, as curated by our APIsecurity.io newsletter editorial team, the report highlights the most common API flaws, from broken input validation and missing authentication to operation-level authorization failures.

Readers will gain actionable insights into the scenarios most likely to expose sensitive data, enabling them to prioritize defenses and governance across their API landscape. Looking ahead to 2026, the report explores the risks and the opportunities presented by APIs in an AI-driven ecosystem, emphasizing the importance of high-quality, consistent APIs for emerging AI agents and LLMs.

Key Insights

Broken Input Validation (28%)
The most common category of API flaws, covering injection, mass assignment, and path traversal.

BOLA & BFLA Failures
Broken authorization remains a leading API security risk, exposing sensitive resources.

Missing Authentication (17%)
The most frequently reported vulnerability in 2025, highlighting gaps in API access controls.

Alignment with OWASP Risks
Our four most prevalent flaws correspond to four of the top five OWASP API risks.

The "Trusted Client" Fallacy
Evidence that API teams often rely on frontend clients to enforce security.

Few Shadow API Exploits
Breaches or vulnerabilities in unmanaged or undocumented APIs are uncommon.

Contents

Introduction	2
Report Overview	4
Categorizing API Vulnerabilities	5
Missing Authentication Is a Top API Vulnerability	9
Broken Authorization Exposes Critical API Risks	11
Malicious Input Drives Most API Vulnerabilities	13
Preventing Unauthorized Data Exposure in APIs	15
Misconfiguration Risks Beyond the API Code	17
Inventory Management for API Security Coverage	19
Client-Side Security Without API-Side Enforcement	20
AI will change the Risk Model for APIs in 2026	22
About 42Crunch	24
API Vulnerability Encyclopedia	25
Categories of AI attacks	29

Report Overview

This report provides essential insights into how modern systems are being exposed through mistakes in API development. We cut through the industry noise by focusing exclusively on 200 real-world API vulnerabilities and exploits from the APISecurity.io newsletter coverage between 2024 and 2025. The cases come from a diverse range of sources, including independent researchers, security firms, and bug bounty programs. These experts are uniquely positioned to uncover the most crucial and valuable vulnerabilities across a range of different industries.

Each case in the report underwent a rigorous manual review by our team of security experts to ensure credible evidence of a production vulnerability or exploit, complete with enough detail to understand its root cause.

The analysis is presented for both a security audience prioritizing risk, and a developer audience focusing on improving API development. We translate lessons learned into concrete, actionable steps that your API teams can take to prevent similar vulnerabilities in 2026.

About APISecurity.io

APISecurity.io is a research and education hub dedicated to highlighting API vulnerabilities, emerging threats, and best practices for secure API development. Founded in 2018 by 42Crunch, a leader in API security, it provides developers and security teams a single source for reliable, up-to-date insights on API security. The biweekly APISecurity.io newsletter has become a widely followed digest that highlights newly discovered API vulnerabilities, breaches, and guidance for securing modern API ecosystems.

Sources

The research team tracks over 100 independent sources spanning a diverse range of perspectives and expertise on API security. These include:

- **Vulnerability databases and security advisories.**
- **Bug bounty programs** reporting on confirmed issues from live applications.
- **Independent experts and security firms** publishing detailed technical analyses.
- **Cybersecurity news outlets** providing news, incident reporting and security trends.

By combining these varied sources of API vulnerability news, the team aims to reflect a broad and accurate picture of the API security landscape, grounded in verified, real-world cases.

Security Firms

armis.com/research
assetnote.io/resources/research
binarysecurity.no
blog.sekoia.io
blogapp.bitdefender.com
blogs/spiderlabs-blog
cloudsek.com/blog
clutch.security/blog
cyble.com/blog
eclypsium.com
evasec.io
forescout.com/resources
fortbridge.co.uk
horizon3.ai
insinuator.net
labs.infoguard.ch
labs.watchtowr.com
logicaltrust.net
patchstack.com
pretera.com
projectdiscovery.io/blog
pwndefend.com
rapid7.com/blog
redteam-pentesting.de
sba-research.org
slcyber.io/research-center
ssd-disclosure.com
techcrunch.com
tenable.com
troyhunt.com
unit42.paloaltonetworks.com
vicone.com/blog/
wiz.io/blog
wordfence.com
zeropath.com/blog

Independent Researchers

abrahack.com
alexschapiro.com
aseem-shrey.medium.com
blog.soohyun.tech
bobdahacker.com
dirkjanm.io
dreyand.rs
eaton-works.com
hiddendom.medium.com
ian.sh
lepoch.at
loopsec.medium.com
medium.com/@attias.dor
medium.com/@impratikdabhi
samcurry.net
sirleeroyjenkins.medium.com
site/zhiniangpeng/blogs
slowmist.medium.com
slugsec.ucsc.edu

Vulnerability Advisories

cisa.gov
exploit-db.com
ftc.gov
github.com
nvd.nist.gov
sec.cloudapps.cisco.com
securitylab.github.com
zerodayinitiative.com

Bug Bounty Programs

hackerone.com
huntr.com

News Outlets

bleepingcomputer.com
computerweekly.com
cyberpress.org
cybersecuritynews.com
darkreading.com
databreachtoday.co.uk
gbhackers.com
securityonline.info
securityweek.com
techradar.com
thehackernews.com
thenewstack.io
theregister.com

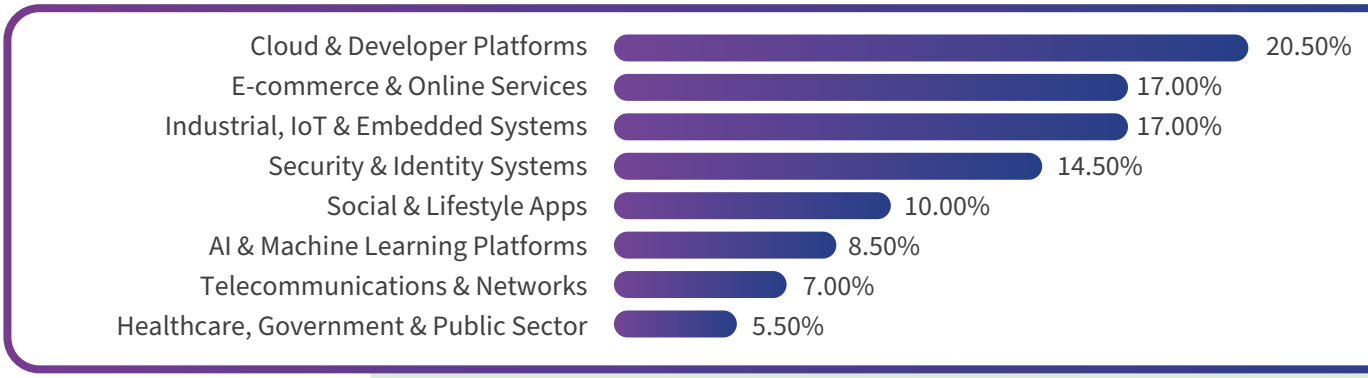
APISecurity.io vulnerability news sources (partial list)

Categorizing API Vulnerabilities

Our objective is to help API teams learn from real world cases and understand the underlying mistakes in development, to build APIs in a way that prevents the most common vulnerabilities.

Industry Coverage

In our analysis, we found that the types and causes of API vulnerabilities are broadly consistent across industries, and from large enterprise platforms to small open-source projects. This means that lessons learned from one industry or project are generally applicable to any API team. The chart below shows the distribution of vulnerabilities in our report by industry, highlighting that API security risks are widespread and not confined to a particular sector.



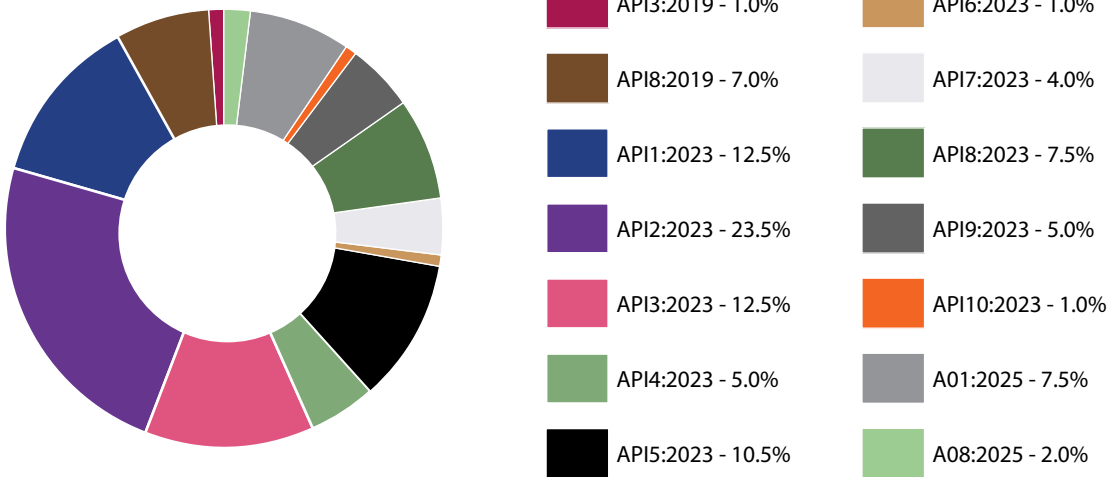
APISecurity.io Vulnerabilities 2024-2025 - Distribution by Industry & Technology

The Security Perspective: Vulnerabilities and Risks

Most vulnerability reports, whether compiled from real-world cases or generated by security tools, organize findings around vulnerability types (e.g. SQL injection, SSRF, XSS) or established risk categories like the OWASP Top 10 lists. These classifications are essential for security teams conducting threat modeling, risk assessments, and compliance reporting.

The chart below shows the real world vulnerabilities included in our report, organized according to the OWASP Top 10 API Security Risks (2023). Our four most frequently reported API vulnerabilities from 2024 to 2025 all fall within the top five OWASP API vulnerabilities. This demonstrates an alignment between the vulnerabilities reported by sources in the field and OWASP's ranking. Those are:

1. API1:2023 Broken Object Level Authorization (BOLA)
2. API2:2023 Broken Authentication
3. API3:2023 Broken Object Property Level Authorization (BOPLA)
4. API5:2023 Broken Function Level Authorization (BFLA)



APISecurity.io Vulnerabilities 2024-2025 - OWASP Categories

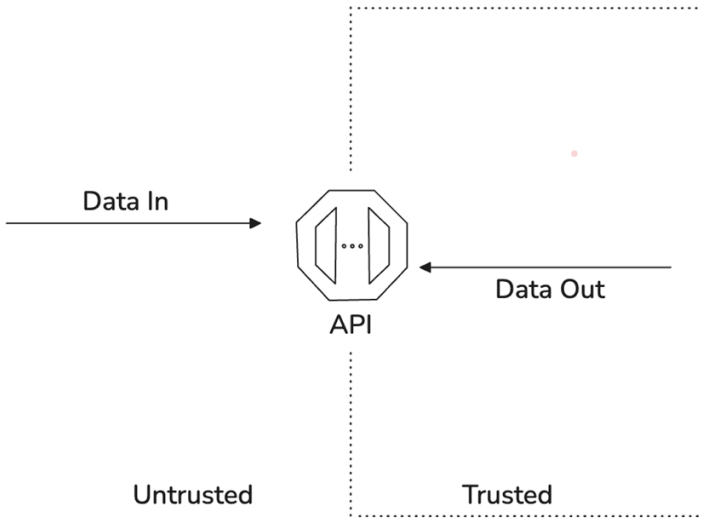
The OWASP Top 10 API list does not cover every possible API vulnerability. To achieve full OWASP coverage of our findings we had to look beyond the 2023 API list. Notably, injection attacks were prevalent in our reporting. While these vulnerabilities are not represented in the 2023 API Top 10, they were included in the 2019 API list (API8:2019).

Similarly, we identified numerous cases of APIs with path traversal flaws. This type of vulnerability does not appear in the OWASP API Top 10 lists, to the best of our knowledge. However, the OWASP documentation for Broken Access Control (A01:2025) references CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'), which we used to categorize these cases.

The Developer Perspective: Responsibility and Prevention

For development teams, we present the same real-world cases, but organized around root causes and clear areas of responsibility for API development. This approach helps teams to implement their APIs according to best practices, making it easier to leverage insights from the report into day to day development practices and prevent vulnerabilities at the source.

So what are the areas of responsibility for secure API development? API development is a discipline in its own right, with distinct considerations and priorities. The main purpose of an API is to expose functionality or data across a boundary, often to a different zone of trust, in a controlled and predictable way (the "I" in API). APIs are also inherently data rich, handling payloads and metadata of varying size, structure, and sensitivity.



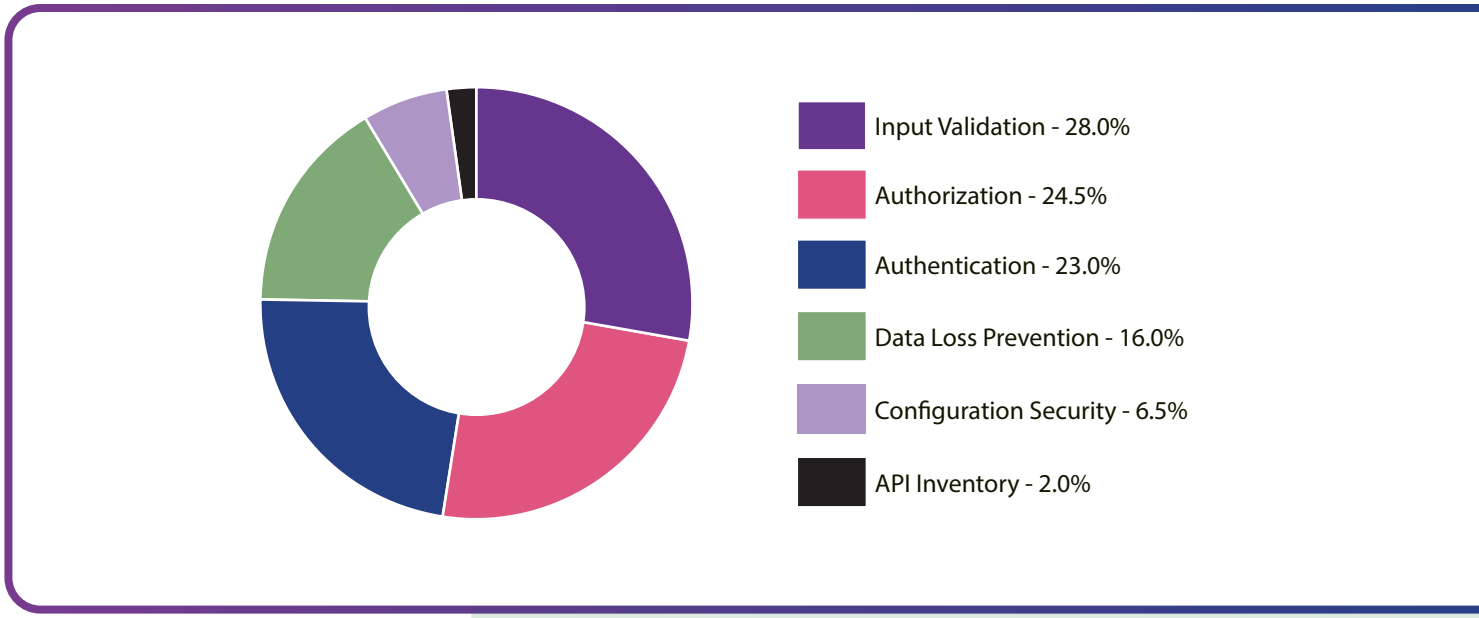
Implementing the "I" in API

These two characteristics, the interface boundary and the data centric nature of APIs, set APIs apart from other areas of software development. Ensuring this boundary is implemented securely is therefore a core responsibility of the API development team. Doing so requires an understanding and focus on the challenges and practices unique to API development, similar to how developers working on real time systems or functional safety code must adopt specialized concerns and practices unique to those domains.

Based on this perspective, we can organize all reported vulnerabilities into categories that correspond to key areas of responsibility for secure API development. These areas are:

- Authentication
- Authorization
- Input Validation
- Data Loss Prevention
- Configuration Security
- API Inventory Management

The chart below organizes the vulnerabilities in our report according to these areas, highlighting where many production-level risks originate and where development teams can focus their efforts to reduce API security exposure.



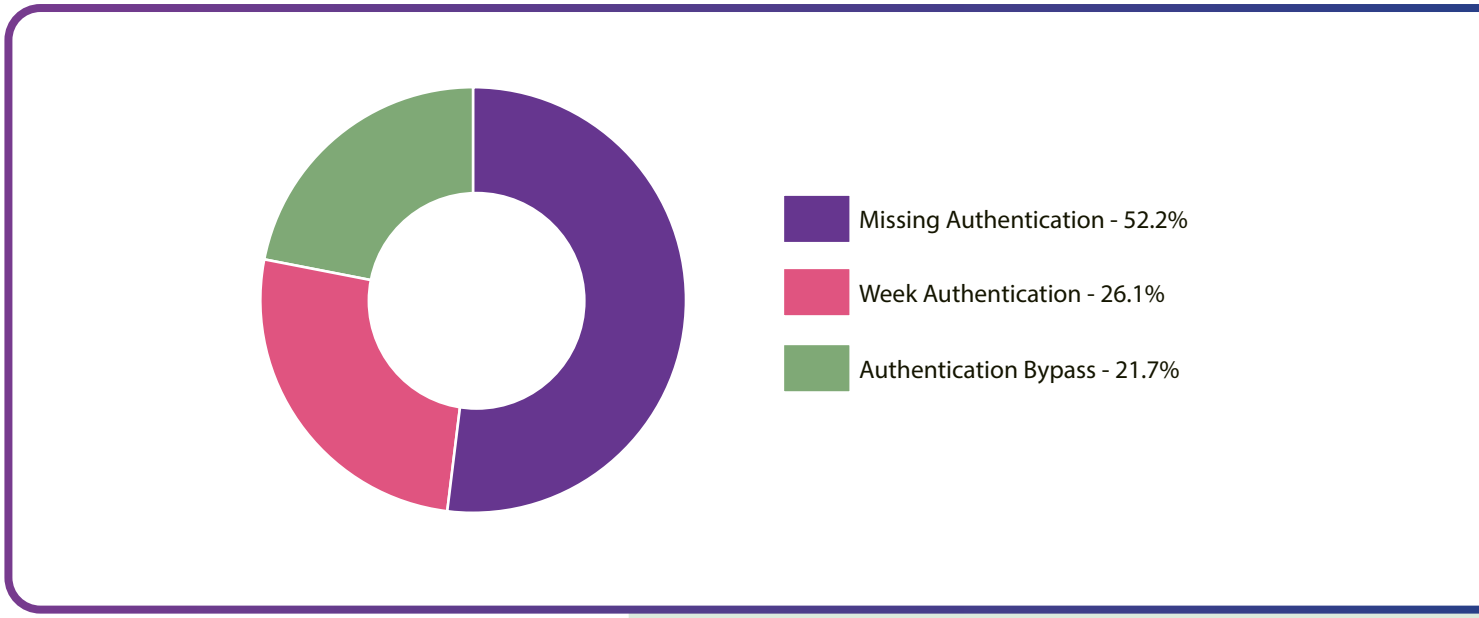
APISecurity.io Vulnerabilities 2024-2025 - API Development Responsibility

Missing Authentication Is a Top API Vulnerability

The most frequent API vulnerability across all categories in 2025 was **Missing Authentication**. This occurs when an API fails to enforce authentication for sensitive operations, allowing anonymous users to access data or perform actions that should be restricted to authenticated users.

Security Risk

The chart below shows the distribution of reported API vulnerabilities related to API authentication.



API Authentication Vulnerabilities 2024-2025 APISecurity.io

Several cases in 2025 highlight the real-world impact of missing authentication. In March, patient data in the UK healthcare system was reportedly exposed through an unauthenticated API. In August, we covered a vulnerability at Intel where a getAccessToken API endpoint returned valid access tokens without requiring any authentication.

Our research also found a recurring pattern across many missing authentication cases. Client-side authentication controls were present, while server-side enforcement was absent. This reflects a common misconception in API development that authentication can be safely delegated to the client application. In practice, APIs must assume they will be called directly by attackers, not just through trusted clients.

Weak authentication vulnerabilities extend beyond missing authentication. They include weaknesses in identity verification mechanisms such as JWT token manipulation, insecure authentication methods like Basic Authentication, credentials passed in cleartext parameters, and authentication endpoints vulnerable to brute force attacks.

7 ways to avoid **JWT Validation Pitfalls** ▶

We also observed a frequent source of authentication bypass caused by overly broad endpoint exclusion logic. When developers intend to expose public endpoints, authentication checks often rely on partial string matching rather than exact paths. Examples we encountered include:

- `contains("public")`
- `endsWith("/serverInfo")`
- `endsWith("?wsdl")`

These checks are unsafe. An attacker can embed the same substring into the path of a protected endpoint. For example, if /serverInfo is treated as public, a request to /api/admin/serverInfo may also bypass authentication while still reaching a privileged operation.

API Development: Endpoint Authentication

API developers should explicitly document every endpoint and clearly define which endpoints are public, which require authentication, and which security scheme applies. This documentation provides a concrete basis for QA and security teams to test, validate, and monitor authentication controls.

Authentication must always be enforced server-side at the API. Client-side controls do not provide meaningful protection and should never be relied upon to restrict access. APIs should be designed with the assumption that they will be directly probed by threat actors seeking anonymous access.

Developers should adopt strong authentication schemes such as OAuth rather than weaker options like Basic Authentication or static API keys. JWTs must be thoroughly validated, including signature, claims, and expiration. Reasonable rate limits should be applied to all authentication endpoints to reduce the risk of brute force attacks.

When defining exceptions for public endpoints, developers should use exact, full-path matching and avoid partial string comparisons that can be easily abused. This reduces the risk of unintended authentication bypass.

```
        340  +        # Only Authentication, Configuration, ServerInof
        341  +        # endpoints are allowed for Anonymous users
        342  +        # if authentication is required.
336     343          if self.server.manager.is_enabled and \
337          -                not self.path.endswith(('/Authentication',
338          -                                        '/Configuration',
339          -                                        '/ServerInfo')) and \
        344  +                request_endpoint not in \
        345  +                ['Authentication', 'Configuration', 'ServerInfo'] and \
340     346                  not self.auth_session:
```

Fixing Authentication Bypass Vulnerability, Source - Github

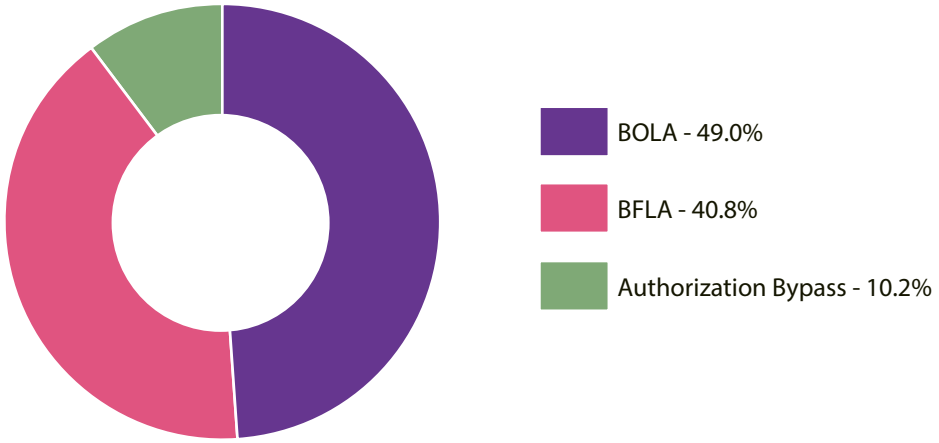
Broken Authorization Exposes Critical API Risks

APIs often support numerous endpoints, and each endpoint can support multiple operations via HTTP methods such as GET, POST, PUT, and DELETE. This complexity makes fine-grained authorization essential, with Broken Object Level Authorization (BOLA) and Broken Function Level Authorization (BFLA) emerging in our findings as prominent challenges in API development.

Enforcing authorization variants (BOLA, BFLA and BOPLA), protecting authentication endpoints and JWT validation. Presenters: Dr. Philippe De Ryck & Isabelle Mauny ▶

Security Risk

The chart below shows the distribution of reported API vulnerabilities related to API authorization.



API Authorization Vulnerabilities 2024-2025 APISecurity.io

We saw many cases of BOLA in 2025. This has become a go-to tactic for API hackers. In one case, a researcher discovered a BOLA vulnerability in India's Goods and Services Tax (GST) website. By manipulating API requests, users were able to access the tax information of private companies that should not have been accessible.

Most of the Broken Function Level Authorization (BFLA) cases we covered involved basic errors in role-based access control at the API operation level. These occurred when functions intended for privileged users were accessible to roles with less permissions. One example of BFLA was uncovered on Securden's Privileged Account Management platform. A sensitive password backup function was available via API to users who were not administrators.

API Development: Endpoint Authorization

Authorization checks for individual resources are usually too fine-grained to offload to centralized platforms like API gateways or IAM products. So the responsibility will often sit with API developers to implement the proper checks at the API endpoint. These authorization checks should verify:

- Does the entity making the request have ownership of, or permission to access, the **object** or resource being requested? The resource will often be accessed by a unique ID (e.g. bookingID, order_id, customer_id).
- Is the entity making the request authorized to perform the requested function (e.g. register on a privileged domain, DELETE or UPDATE a resource)?

```
// Critical authorization check (BOLA prevention):
if (claims.userId !== resourceOwnerId) {
  return res.status(403).json({ error: "Unauthorized access" });
}
```

Checking object-level authorization

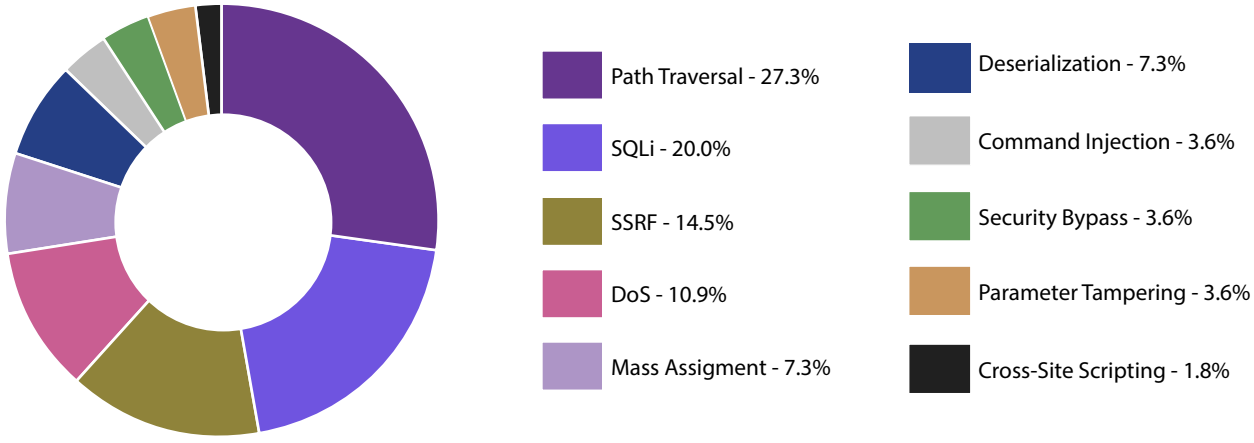
Developers should take a zero trust approach when coding APIs. Never assume client-side controls provide meaningful protection. Always design APIs with the assumption that they will be directly probed by threat actors attempting to gain unauthorized access to resources and operations.

Malicious Input Drives Most API Vulnerabilities

The data-rich nature of APIs makes them highly susceptible to attacks involving malicious input. This category represents the largest share of reported cases (28%), and includes common attack patterns such as path traversal, SQL injection (SQLi), Server-Side Request Forgery (SSRF), and Mass Assignment.

Security Risk

The chart below shows the distribution of reported API vulnerabilities exposed by different forms of malicious input.



API Input Validation Flaws 2024-2025 APISecurity.io

Path traversal vulnerabilities were observed in several platform APIs that handle file access, including Cisco Identity Services Engine, Trellix Enterprise Security Manager, and LG smart TVs. In each case, the APIs failed to properly sanitize input containing some form of the path traversal pattern "../", enabling attackers to navigate directories and access unauthorized resources.

Similarly, APIs exposed to SQLi and SSRF allowed malicious input to bypass the API layer and exploit weaknesses in code that constructs database queries or generates URLs from user-supplied data. Some reported cases include an SQLi vulnerability in F5's BigIP Next platform API, and an SSRF vulnerability reported in the API of the Invoice Ninja software.

API Development: Input Validation

Developers are best positioned to define exactly what input their API expects, making them the most reliable source for specifying intended API behavior. By strictly whitelisting valid input, they not only ensure the API functions correctly but also prevent malicious payloads from crossing the API boundary and reaching deeper vulnerabilities. This approach empowers developers to contribute meaningfully to security and threat prevention, without requiring deep security expertise.

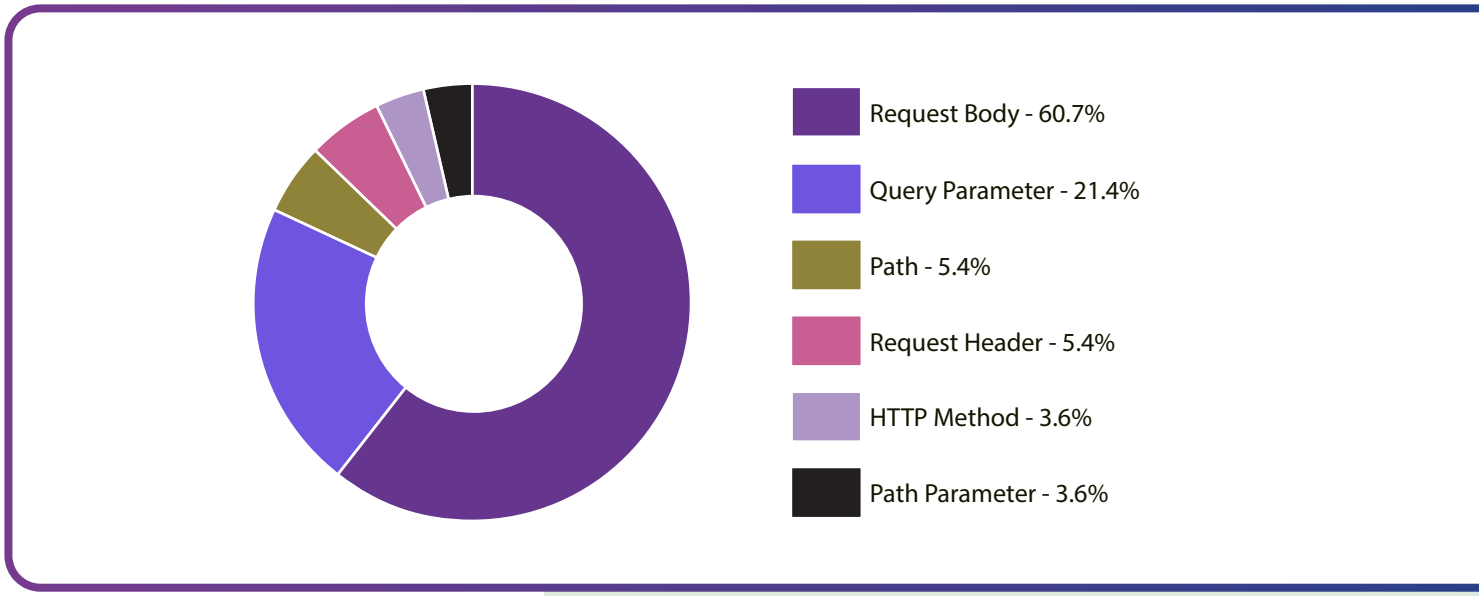
Strict schema enforcement ensures that:

- Only required or explicitly allowed headers, parameters and properties are accepted
- All data conforms to expected formats, value ranges, and size or complexity constraints, and is well formed and valid for the service.

Beyond schema enforcement, other user-controlled input should also be constrained:

- API URLs and endpoint paths should be fully whitelisted to expose only intended routes.
- HTTP methods should be restricted to those necessary for the API's intended functionality.

Mapping input validation vulnerabilities along the API's surface reveals where developers can gain the most impact from enforcing controls, turning input validation into a strong first line of defense.



Distribution of Malicious Input Vulnerabilities by API Attack Surface

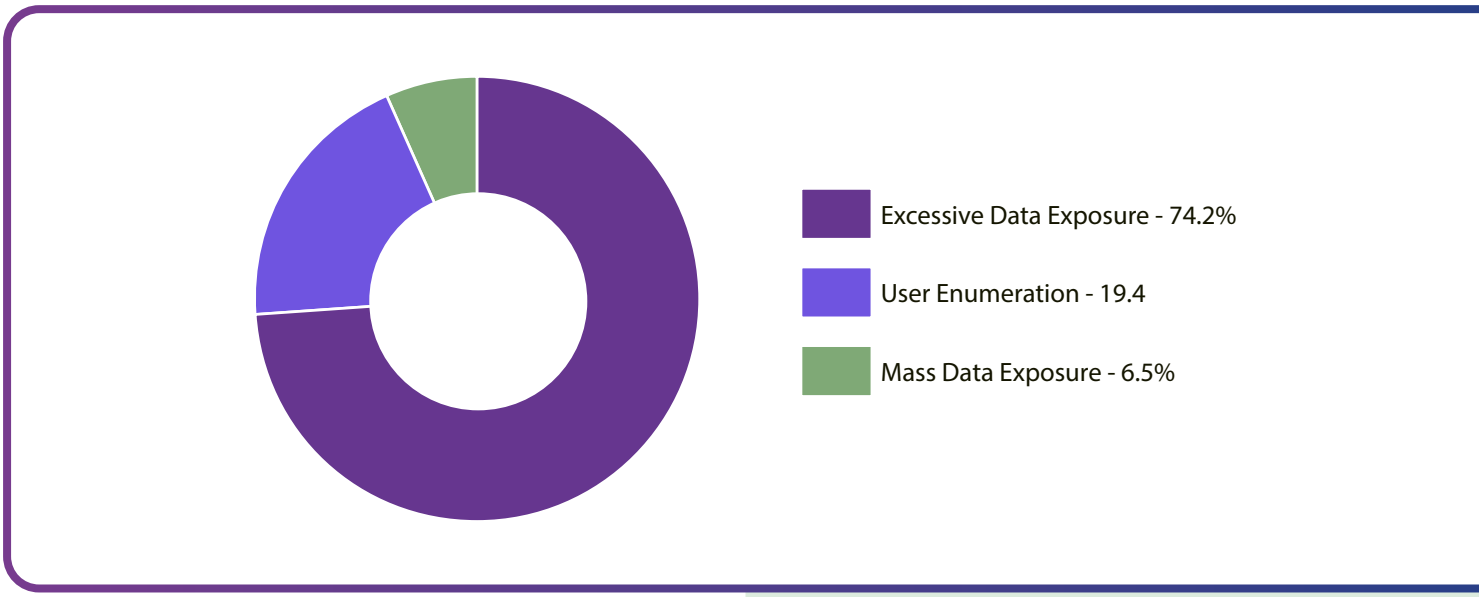
This shows that for more than 80% of the reported API vulnerabilities tied to malicious input, developers can have the greatest impact on security by enforcing strong input validation on request bodies and query parameters.

Preventing Unauthorized Data Exposure in APIs

Data exposure remains one of the most persistent and high-impact flaws in API development. APIs routinely handle rich data structures, with payloads and metadata that evolve over time. Without proper controls, APIs can begin to expose unexpected or unauthorized data at any time.

Security Risk

The chart below shows the distribution of reported API vulnerabilities related to API data loss prevention.



API DLP Vulnerabilities 2024-2025 APISecurity.io

The majority of API data leaks involved the exposure of unexpected or unauthorized properties in API responses. This aligns with OWASP API3:2023 Broken Object Property Level Authorization, although the term **Excessive Data Exposure** from the 2019 list more clearly describes the underlying issue.

Several real world cases highlight the impact. In February 2024, the social media company Spoutible exposed excessive user data through a leaky API. In May 2025, a vulnerability in Volkswagen's connected vehicle system returned API responses containing unexpected data, including plaintext secrets and passwords. In October, a registration portal for Formula 1 drivers exposed additional properties in API responses, providing enough context for researchers to identify a privilege escalation vulnerability.

Another common form of data loss is **user enumeration**, where APIs inadvertently reveal information about registered users. For example, a registration API might indicate whether a supplied email is already registered, unintentionally exposing private information. We observed multiple enumeration cases in API error responses. In one case, a failed login attempt on a Life360 Android account returned the first name and phone number of the target user.

When APIs vulnerable to user enumeration also lack rate-limiting, this can escalate into mass data scraping attacks, as seen in breaches affecting companies like Trello and Authy.

API Development: Data Loss Prevention

Clearly define the exact data each API endpoint is allowed to return, and enforce these definitions consistently throughout the API lifecycle. As APIs evolve, failing to update and validate these definitions can create hidden opportunities for data leaks that are difficult to detect and easy to exploit.

Enforce strict schema validation on all API responses, and do not allow any properties beyond what is explicitly defined. Apply the same rigor to error responses to maintain control over all data flows.

```yaml
responses:
  "201":
    description: OK
    content:
      application/json:
        schema:
          type: object
          properties:
            message:
              $ref: "#/components/schemas/ResponseMessage"
            accountId:
              $ref: "#/components/schemas/AccountId"
          additionalProperties: false
```

Defining an API's response properties

Limit the size and scope of data returned to only what is necessary and authorized for each request.

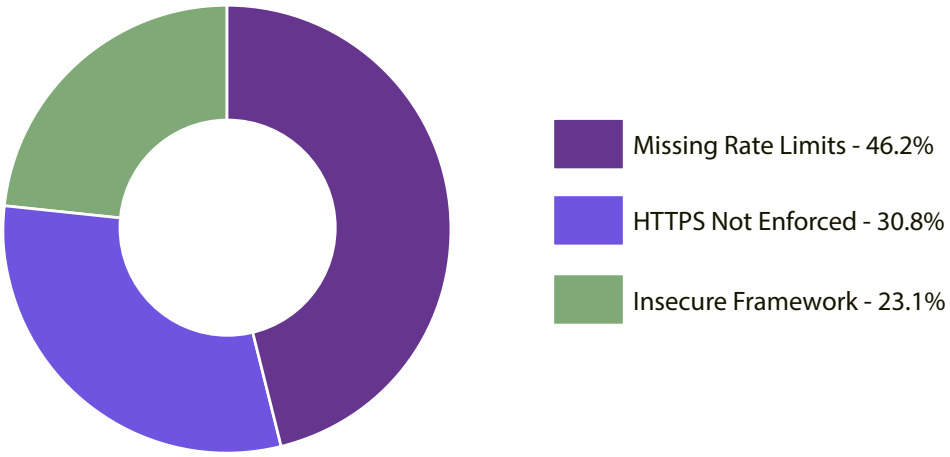
Never expose information about other users unless it is an explicit part of the service. Pay close attention to error messages and user lookup operations to prevent inadvertent user enumeration.

Misconfiguration Risks Beyond the API Code

While most reported API vulnerabilities stem from errors or omissions in API code, a number of reported issues were caused by misconfigurations that directly impact API behavior and weaken security controls.

Security Risk

The chart below shows the distribution of reported API vulnerabilities related to configuration security.



API Configuration Security Vulnerabilities 2024-2025 APISecurity.io

We documented multiple cases of large-scale data scraping through platform APIs where rate limiting was not enforced. In 2024, Trello and Authy made headlines after threat actors exfiltrated millions of user records. In 2025, similar weaknesses were identified in APIs operated by WhatsApp and the airline booking website AveloAir.

In some cases, missing rate limits were compounded by user enumeration flaws, allowing APIs to verify the existence of millions of registered users. These verified user lists can be highly valuable for automating credential stuffing and spear phishing attacks. In other cases, platforms intentionally expose user lookup APIs, commonly used by social media services to help users discover contacts. Attackers routinely abuse these features to compile large-scale databases of registered users.

Failure to enforce HTTPS represents another configuration vulnerability. In October 2024, cybersecurity firm VicOne reported that several vehicle tracking system providers in Europe and the United States were exposing login endpoints without Transport Layer Security. In January 2024, the US Federal Trade Commission took action against GoDaddy, citing poor security practices that included failing to secure APIs with HTTPS.

Modern API development also relies heavily on frameworks and third-party libraries, which can introduce vulnerabilities if misconfigured or deployed with insecure defaults. In December 2024, Volkswagen's Cariad software exposed large volumes of sensitive driver data through an API misconfiguration in the Spring framework, a widely used Java platform. In a separate 2024 case, a bug bounty researcher discovered a Django-based API operated by telecommunications group MTN that had been deployed in debug mode, leaking sensitive information in 404 error responses.

API Development: Configuration Security

Rate limiting is enforced at the API operation level primarily to protect against denial of service and brute force attacks, and to enforce usage limits on sensitive business workflows. Rate limiting can also help reduce the impact of mass enumeration attacks, but it should not be relied on as a complete control and must be tuned to avoid disrupting legitimate users.

Protect all API traffic with strong Transport Layer Security. Implement continuous or scheduled checks against production APIs to detect configuration changes that weaken or disable TLS enforcement.

Review all frameworks and third-party dependencies to ensure secure defaults are enabled before deployment. Use explicit route whitelisting to ensure that only intended API endpoints are exposed to external consumers.

Enforce strict schema validation on all API responses to detect and block unintended data exposure caused by misconfigured frameworks or dependencies.

Inventory Management for API Security Coverage

API inventory management is a foundational component of API governance, providing essential visibility and control over all APIs and ensuring that governance policies are consistently applied.

APIs that are not included in a shared inventory may bypass critical governance processes and security controls, leaving them vulnerable in production. Though, clearly APIs that are documented and tracked also create risk when security testing or protections are missing or incomplete. Governance is about the systematic application of effective security practices, and inventory management is about coverage.

Security Risk

In June 2025, we documented a case involving an API used by Bykea's Android application that was no longer in active use but remained accessible and was vulnerable to Broken Object Level Authorization (BOLA). In a separate case reported in December 2024, analysis of the McDonald's delivery application identified an undocumented route at the root of an API path that returned sensitive data, illustrating a data flow blind spot that could be exploited.

Beyond these cases, our review of publicly available findings from independent API researchers, security firms, and bug bounty programs, experts that routinely conduct API reconnaissance, identified very few documented cases where untracked APIs were discovered or successfully exploited.

Given the level of industry attention on API discovery and so-called hidden APIs, we expanded our review to include historical vendor reports citing extremely high volumes of activity, in some cases billions of requests, attributed to attacks targeting unmanaged APIs. Across these reports, we found no evidence demonstrating that this activity led to confirmed API breaches, the exposure of vulnerabilities, or the discovery of previously unknown, untracked valid API routes.

Large attack volumes are more plausibly explained by opportunistic scanning using automated tools that generate broad sets of generic attack patterns across many endpoints. Such scanning can easily produce billions of requests, even from a single actor, but volume alone is not an indicator of successful API discovery or compromise.

API Development: API Inventory Management

How to Achieve a **Complete API Inventory** ▶

Developers play a critical role in accurately documenting and tracking the API estate as APIs and endpoints are created, updated, and retired. By clearly specifying for each API operation:

- which endpoints and operations are explicitly supported
- what input data it should accept or require
- whether it is public or requires authentication and the security scheme used
- what data it should expose to API consumers

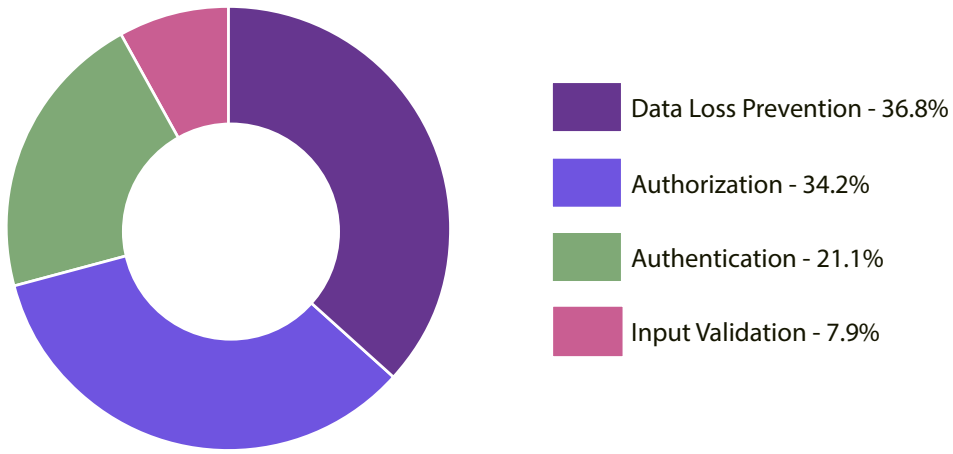
developers provide a development-driven source of truth that benefits the entire organization. This ensures company-wide adherence to API security policies, supports full API test coverage, and provides a reliable foundation for runtime monitoring and anomaly detection.

Client-Side Security Without API-Side Enforcement

Our research uncovered a recurring issue we call a client-side security mindset, where API developers rely on client applications to enforce security instead of implementing controls at the API layer. This pattern appeared in roughly 20% of the API cases we analyzed. It reflects a breakdown in DevSecOps collaboration between frontend, backend, and security teams and a lack of awareness that attackers can easily bypass client-side protections to exploit unprotected APIs.

Security Risk

The chart below shows the share of API vulnerabilities in which client-side protections were in place, but corresponding security controls were missing at the API layer.



Distribution of API vulnerabilities with client side protections

Data Loss Prevention

We observed cases where APIs send excessive data to clients and rely on the client application to filter what is displayed to the end user. This practice was widespread in vulnerability reports on dating and social media applications, where API traffic exposed far more information, including sensitive user data, than the client actually showed.

In August 2025, a researcher highlighted a similar issue in APIs powering Intel's employee website, which returned far more employee data than necessary for the consuming application. In each case it was trivial for the researcher to access the API traffic and observe the excess data leaked by the API developer back to the client.

API Authentication

In other cases we covered, APIs do not enforce authentication because developers may assume users are already logged in through the client application and so do not require authentication at the API layer. For instance in February 2025, vulnerabilities were reported in the India Post Office website. Although users had to log in to access the site, the APIs serving sensitive user data to the website did not require or enforce access tokens. Authentication was effectively enforced only on the client side, leaving the API itself unprotected.

API Development: Server Side Enforcement

Treat APIs as the primary gatekeepers of sensitive data and operations. Enforce authentication, authorization, and data access policies **at the API level**, independent of client-side controls. Never assume that the client application alone can enforce security.

Align frontend and backend teams to ensure responsibilities for data protection and access control are clearly defined and understood.

AI will change the Risk Model for APIs in 2026

Several observations from this report have important implications for how AI agents will consume APIs and for the risks that emerge from this new mode of interaction.

First, our analysis is based on more than 200 real-world cases of API vulnerabilities that reached production systems. The vulnerabilities stem largely from common mistakes in the APIs implementation or configuration, and we note the same vulnerabilities are consistent across many different industries and project types. The implication is that the API vulnerabilities covered in this report are likely present to some degree in API deployments everywhere.

Second, many of the vulnerabilities documented in this report were present in production for extended periods of time, in some cases for years. These issues did not surface through normal operation of the APIs. Instead, they were uncovered only when experienced researchers deliberately examined API behavior in depth. This typically involved analyzing API traffic, using experience to spot signs of potential vulnerabilities, and probing endpoints through systematic trial and error in order to understand the full range of possible behaviors and exploits.

This raises an important question. Why were these vulnerabilities not discovered earlier?

The answer lies in how APIs are typically consumed today. Most API clients, including mobile applications, web applications, SaaS platforms, and IoT devices, are programmed to interact with APIs in a predefined way. They do not adapt their behavior based on unexpected API responses or changing security conditions.

For example, if an API begins returning excess data in a response, as seen in several data loss prevention cases in this report, existing clients will continue to process only the fields they were programmed to use. The extra data is ignored and the issue may go unnoticed unless someone explicitly inspects the traffic. Similarly, if an API stops enforcing authentication due to a configuration change or a regression, a platform calling that API will typically continue sending credentials as before. It does not learn from or reason about changes in an APIs behavior.

AI agents fundamentally alter this dynamic

Unlike traditional clients, AI agents are not preprogrammed to interact with APIs in a fixed way. They operate more like human researchers, learning over time how an API behaves by observing responses, testing boundaries, and inferring patterns. In doing so, they naturally incorporate the full operational context of an API, including undocumented behavior, inconsistent security controls, and latent vulnerabilities.

Critically, when AI agents encounter unexpected behavior or exploitable conditions, they do not ethically report those issues back to the organization as the researchers do. Instead, they may surface the issues to users and customers through prompts, recommendations, or automated actions that reveal how services and resources can be used or misused through vulnerable APIs.

This represents a major shift in API risk. AI enablement increases the likelihood that latent API vulnerabilities will be exercised in production. As a result, APIs must be as well-defined, constrained, and predictable as possible before they are exposed in production environments where AI agents can interact with them. This reinforces the importance of API governance as an integral part of the development lifecycle, to build high quality and secure APIs.

Implementing Guardrails when using AI Coding Assistants ▶

The developer-focused approach of this report is intentional. By presenting real-world vulnerability cases in the context of mistakes made in API implementation, the goal is to help development teams internalize these lessons and incorporate them into everyday governance practices.

This proactive, preventative approach, building secure APIs by design, is essential for reducing risk in an environment where AI agents are becoming first-class consumers of enterprise APIs.

Yet it is equally important to be able to extend the design-time governance to a runtime enforcement model, particularly when considering the scalability, adaptability and velocity of AI-driven attacks.

Extending from Design-Time Governance to Runtime Enforcement

42Crunch addresses this challenge by extending developer-driven API governance into runtime enforcement. APIs are validated and governed against OpenAPI contracts during development, and those same contracts are enforced in production to ensure APIs behave exactly as designed. This approach allows 42Crunch to block many AI-driven attacks in real time, including schema abuse, exploratory probing, privilege inference, and response mining—before AI agents can incorporate those behaviors into their learning loops.

Predictability as a Security Control

In the AI era, predictability becomes a security control. By making APIs deterministic, constrained, and contract-driven at runtime, 42Crunch prevents AI agents from inferring new capabilities or escalating access through experimentation. This combination of proactive governance and real-time enforcement enables organizations to safely expose APIs to AI-enabled systems while maintaining control, reducing risk, and preserving developer velocity.

About 42Crunch

42Crunch provides a standardized approach to API security that automates the enforcement of security and governance across distributed development, security, and platform teams. As organizations accelerate their adoption of AI, the API risk landscape is expanding rapidly. 42Crunch helps enterprises address these challenges by ensuring APIs are well-defined, governed, and enforceable across the full lifecycle, from design to runtime. By turning API contracts into active security controls, the 42Crunch API Security Platform enables organizations to reduce risk, prevent abuse, and safely expose APIs to AI-enabled systems at scale.

The 42Crunch API Security Platform delivers API security testing, governance, and runtime protection and is used by Fortune 500 enterprises and more than 2 million developers worldwide.

Additional Reading

DAST v API Contract Testing

DAST is failing your APIs. Go beyond DAST with API Security-by-design, the smarter way to protect your APIs ▶

Behavior Monitoring is Not Enough

Why using machine learning alone falls short for API security and why 'Secure by Design' is essential ▶

How to Build an API Inventory

How to Create a Future-proof Living API Inventory and avoid vendor lock-in ▶

Contact 42Crunch

sales@42crunch.com ▶
42crunch.com ▶

API Vulnerability Encyclopedia

This encyclopedia documents each type of API vulnerability covered in this report. The descriptions are derived from real world cases.

Authentication

Purpose: Ensure that API endpoints requiring authentication are properly identified, documented, and protected, so that only verified callers can access them.

Scope: Identifying which endpoints require authentication, enforcing credentials or tokens, session management, credential validation (API keys, JWTs, passwords), ensuring authentication is applied correctly and securely where needed.

Missing Authentication

APIs that do not enforce authentication for sensitive operations, allowing anonymous or unauthenticated users to perform actions that should be restricted to authenticated users.

Weak Authentication

Weaknesses in authentication mechanisms that allow attackers to compromise user accounts. This includes JWT token manipulation (modifying tokens to impersonate other users), weak password policies (APIs that don't enforce password complexity rules enforced by the UI), insecure authentication methods (using basic authentication with plaintext credentials), cleartext credentials in URLs or parameters, and lack of rate limiting on authentication endpoints enabling brute force attacks.

Authentication Bypass

Vulnerabilities that allow attackers to bypass authentication mechanisms entirely. Common methods include path manipulation (changing URL case, adding fake endings), header manipulation (spoofing X-Forwarded-For headers), or exploiting misconfigured authentication checks.

Authorization

Purpose: Control what each authenticated caller is allowed to do, ensuring they can only access permitted resources and perform allowed actions.

Scope: Object-level, role-level, scope-level, and function-level access control, covering all mechanisms that enforce least privilege and prevent unauthorized operations.

Broken Object Level Authorization

APIs that fail to verify whether a user has permission to access a specific object. Attackers can access resources belonging to other users by manipulating identifiers in API requests, such as changing user IDs, account numbers, or resource identifiers.

Broken Function Level Authorization

APIs that fail to verify whether a user has permission to perform a specific function. Regular users may access administrative endpoints, perform privileged operations, or execute functions reserved for administrators or higher privilege levels.

Authorization Bypass

Vulnerabilities that allow attackers to bypass authorization checks through path manipulation, parameter tampering, misconfigured access control rules, or business logic flaws that enable unauthorized access to restricted functions or resources.

Input Validation

Purpose: Ensure all input provided by clients is properly structured, well-formed, and safe before being used by the API or business logic.

Scope: Path parameters, query parameters, headers, and request body payloads; content type and schema validation; data type enforcement; strong typing; and adherence to the API contract.

Path Traversal

Vulnerabilities that allow attackers to access files or directories outside the intended directory structure by manipulating file paths in API requests. Attackers use sequences like "../" or encoded variations to traverse the file system and access unauthorized resources.

SQL Injection

Vulnerabilities where user input is not properly sanitized before being used in database queries. Attackers can inject malicious SQL code through API parameters to read, modify, or delete database contents, potentially gaining unauthorized access to sensitive data.

Server Side Request Forgery

Vulnerabilities where APIs accept user-supplied URLs and make requests to those URLs without proper validation. Attackers can force the server to make requests to internal resources, cloud metadata services, or perform network reconnaissance.

Denial of Service

Vulnerabilities that allow attackers to cause service unavailability through resource exhaustion. This includes APIs that accept unlimited request sizes, lack rate limiting, perform expensive operations without limits, or return unconstrained response payloads.

Mass Assignment

A vulnerability where APIs accept and process object properties that should not be modifiable by the user. Attackers can send additional fields in request bodies to modify unauthorized properties such as user roles, account privileges, or sensitive configuration settings.

Deserialization

Vulnerabilities where APIs deserialize untrusted data without proper validation, allowing attackers to execute arbitrary code, cause denial of service, or manipulate application logic by providing malicious serialized objects.

Command Injection

Vulnerabilities where user input is not properly sanitized before being used in system commands. Attackers can inject operating system commands through API parameters to execute arbitrary code on the server.

Parameter Tampering

Vulnerabilities where APIs do not properly validate or sanitize parameters, allowing attackers to manipulate query parameters, path parameters, or request body parameters to bypass security controls or access unauthorized resources.

Security Bypass

Vulnerabilities that allow attackers to bypass security controls through various methods such as HTTP method manipulation, header manipulation, or exploiting misconfigured security rules that fail to properly enforce access controls.

Cross-Site Scripting

Vulnerabilities where APIs do not properly sanitize user input that is later reflected in responses, allowing attackers to inject malicious scripts that execute in the context of other users or systems consuming the API.

Data Loss Prevention

Purpose: Prevent sensitive, excessive, or unauthorized data from being returned by the API, intentionally or inadvertently.

Scope: Excessive data exposure (unauthorized fields), sensitive data in authorized fields, error messages leaking internal information, over-returned collections, exposure of PII/PHI, and response minimization or redaction.

Excessive Data Exposure

APIs that return more data than necessary in responses, exposing sensitive information that should be filtered or redacted. This includes returning full user objects when only specific fields are needed.

User Enumeration

Vulnerabilities that allow attackers to determine whether a username, email, or account identifier exists in the system. APIs may reveal this information through different error messages, response times, or by returning different data for valid versus invalid users.

Mass Data Exposure

APIs that return unlimited or excessively large response payloads without pagination or size limits. This can lead to resource exhaustion, denial of service, excessive bandwidth consumption, and potential data exposure.

Configuration Security

Purpose: Protect the API interface and runtime environment by enforcing safe protocol behavior and secure defaults.

Scope: CORS configuration, TLS settings, required headers, rate limiting, response headers, and server-side default configuration risks.

Missing Rate Limits

APIs that lack rate limiting controls, allowing attackers to make unlimited requests. This enables brute force attacks, credential stuffing, denial of service, and abuse of business logic functions like account creation or password resets.

HTTPS Not Enforced

APIs that allow unencrypted HTTP connections, exposing sensitive data such as credentials and tokens to interception and manipulation.

Insecure Framework

APIs built on frameworks with insecure defaults or misconfigurations, such as debug mode enabled or endpoints exposed without authentication.

API Inventory Management

Purpose: Ensure all APIs, endpoints, and supported operations are properly documented, discoverable, and controlled, so developers maintain visibility and reduce unintended exposure.

Scope: HTTP method whitelisting, URL/endpoint whitelisting, management of deprecated or legacy APIs, zombie endpoints (unused but reachable), and completeness of documentation.

API Not Retired

APIs that remain active after they are no longer used, exposing outdated functionality and increasing the attack surface unnecessarily.

Data Flow Blindspot

APIs where the movement of sensitive data is not fully tracked, creating gaps that can lead to unmonitored exposure or leaks.

Categories of AI attacks

Just as we identified above with APIs, there is a set of familiar and new attacks specific to the world of agentic AI from which companies will need to protect their resources. We itemize and describe these attacks here.

AI Attack Category & Individual Vectors	Attack Type
1. Prompt Driven API Misuse (AI → API)	LMs can be manipulated into making API calls they were never intended to make.
Prompt Injection → Unauthorized API Calls	Attacker convinces the model to call privileged endpoints.
Induced Parameter Manipulation	Model is tricked into sending dangerous parameters (SQL strings, oversized payloads).
Function Calling Abuse	Attacker forces the model to select a sensitive function via schema manipulation.
Workflow Escalation	Multi-step agents are manipulated to chain calls to escalate privileges.
2. Hallucination Driven API Risk (AI → API)	LLMs hallucinate—and hallucinations can become API calls.
Hallucinated Endpoints	Model invents endpoints that don't exist, causing error storms.
Hallucinated Parameters	Model fabricates fields that bypass validation.
Hallucinated Identities	Model assumes roles or tokens that don't exist.
3. Data Leakage Through API Responses (API → AI → User)	AI models amplify leakage because they summarize and expose data.
Over Disclosure via LLM Summaries	API returns sensitive data; LLM summarizes it for the user.
Cross Record Leakage	LLM merges multiple API responses, violating isolation.
PII Rehydration	LLMs infer missing details from masked data.

4. Identity & Authorization Breakdowns (AI ↔ API)	AI agents break traditional assumptions about identity and sessions.
Agent Identity Ambiguity	Confusion over who is calling: user, agent, or model.
Token Over Delegation	LLMs running with broad API keys are exploited.
Session Confusion	LLM mixes multiple user contexts.
Replay via Model Memory	Attackers extract or reuse tokens stored in context.
5. AI Driven Abuse of API Rate Limits & Quotas	AI agents generate traffic at machine speed, leading to exhaustion.
LLM Amplified DDoS	A single prompt triggers thousands of backend calls.
Recursive Agent Loops	Agents stuck in logic loops hammer APIs.
Fan Out Explosions	LLMs call multiple APIs in parallel, overwhelming systems.
6. Model Mediated Injection Attacks (API → AI → API)	APIs return data that manipulates the LLM to trigger further calls.
Response Injection	API returns text that manipulates the LLM.
Schema Injection	API returns JSON that tricks the model into dangerous functions.
7. Supply Chain & Plugin/Tooling Risks (AI ↔ External APIs)	AI agents often call external APIs automatically.
Malicious Third Party Tools	Fake "AI Agents/plugins" that steal tokens.
Dependency Confusion	Attackers register similarly named tools to intercept calls.
External API Injection	External APIs return crafted responses to manipulate the LLM.

8. AI Driven Business Logic Abuse (AI → API)	AI is used to explore and exploit edge cases in business logic.
Automated Discovery of Logic Flaws	LLMs fuzz APIs more intelligently than humans.
Policy Circumvention	LLMs find sequences of calls that bypass rules.
Semantic Abuse	LLM performs harmful but "valid" API actions.