KushoAI

State of API Security 2026: An AI-Native Testing Perspective

Observed security failure patterns from 1.4M AI-driven test executions

1.4M+
Security-Relevant Test Executions

2,600+
Organizations

10
OWASP Categories Mapped

Published by KushoAI - April 2026 (link)

KushoAI

Table of Contents

About KushoAI

KushoAI is building AI-native infrastructure for software maintenance - autonomous housekeeper agents that keep production systems clean, secure, and reliable. They embed within CI/CD pipelines and continuously test, heal, and monitor code so engineering teams can focus on new product work instead of repetitive maintenance.

KushoAI is used by engineering and QA teams across enterprises in fintech, infrastructure, and consulting.

To learn more or request a demo, visit kusho.ai or write to us at hello@kusho.ai.

Methodology

The findings in this report are drawn from telemetry collected across KushoAI's platform between January and December 2025. The dataset covers 1.4 million test executions across 2,616 organizations, spanning early-stage SaaS products, growth-stage companies, and large institutions across industries including fintech, healthcare, e-commerce, and enterprise software.

All data was collected passively through normal platform usage. No surveys were conducted.

Security failure classification maps observable signals in test execution logs to the OWASP API Security Top 10 taxonomy using a combination of rule-based and model-assisted classification. Observable signals include HTTP response codes, assertion outcomes, response schema deviations, and header analysis. Purely functional failures — wrong response values on valid inputs, schema mismatches with no security consequence, incorrect business logic that does not create an exploitable condition — are excluded. Category assignments are directional rather than definitively classified and reflect patterns in execution data rather than confirmed vulnerability assessments.

Organizations are categorized by industry based on self-reported verticals at the time of account creation. The supply chain section draws on public incident reports, CISA advisories, and security community disclosures rather than platform data, except where noted.

All data used in this report is anonymized and aggregated. No organization-level or individual-level data is identifiable in any finding.

How to cite this report

KushoAI. State of API Security 2026: An AI-Native Testing Perspective. April 2026. https://reports.kusho.ai/state-of-api-security-2026

Executive Summary

This report draws on the same dataset behind the **State of Agentic API Testing 2026**: 1.4 million test executions across 2,616 organizations, mapped to the OWASP API Security Top 10 to surface where APIs fail and where test suites fail to look. To our knowledge, it is the largest published analysis of API security failures observed from AI-driven testing activity rather than survey responses or penetration testing engagements.

38% of all security failures are auth and authorization issues. Not because auth is hard to build, but because auth edge cases are systematically undertested. Most suites verify that unauthenticated requests are rejected. Fewer than 30% verify that authenticated requests are correctly scoped. These are not theoretical gaps. They are failures reaching production APIs in active development at organizations across every industry vertical in this dataset.

34% of all test failures in this dataset have a direct security implication. These are not red team findings. They are failures surfaced by automated test suites on APIs in active development — failures the owning teams were already positioned to fix, had the right tests been present.

AI-generated test suites cover 2.7x more OWASP categories than manually authored ones, with the largest gaps in cross-user access probes, privilege escalation checks, and SSRF: exactly the categories manual authors skip most often.

Supply chain attacks now represent the fastest-growing API security threat class, and the current testing toolchain has no coverage of them at all. The incidents documented in Section 6 show what that gap looks like in practice. Closing it will require both new tooling and a broader definition of what API security testing means.

The Security Failure Landscape

Across the 1.4 million API test executions in this dataset, drawn from 2,600+ organizations ranging from early-stage SaaS products to large financial institutions, **34% of all test failures** have a direct security implication. That figure includes any assertion failure involving authentication, authorization, input validation, data exposure, or security configuration, mapped to the OWASP API Security Top 10 taxonomy using a combination of rule-based and model-assisted classification.

Purely functional failures (wrong response values on valid inputs, schema mismatches with no security consequence, incorrect business logic that does not create an exploitable condition) are excluded.

These are not penetration testing findings or red team results. They are failures surfaced by automated test suites running against APIs in active development: the kind of failures that reach production when the right test types are absent.

One in three API test failures, in a dataset of over a million executions, has a direct security implication. The distribution of those failures across OWASP categories is what this section documents.

38%

of all API test failures are auth or authorization related

34%

of all test failures have a direct security implication

Security Failure Categories: Share of All Security-Relevant Failures

Failure Category	OWASP	Share
Auth / Authorization failure	API2,API5	38%
Broken Object Level Authorization	API1	22%
Input validation / injection surface	API3, API8	18%
Excessive data exposure / mass assignment	API3	9%
Rate limiting absent or bypassable	API4	7%
Security misconfiguration	API8	4%
Other / Unclassified	-	2%

The dominance of auth and authorization failures here is not evidence that developers write bad auth code. It is evidence that auth edge cases are not tested. Token validation, scope enforcement, and cross-user isolation tend to be implemented correctly on the primary flow, but the edge cases slip through.

A token with insufficient scope gets accepted because the middleware checks that a token is present and valid, not that it covers the requested operation. That passes every happy-path test. It only fails when a test specifically tries the wrong scope. The dataset reflects this precisely: failures concentrate not in the primary auth path, but in the conditions around it.

OWASP API Top 10: Coverage and Observations

The OWASP API Security Top 10 provides the most widely adopted taxonomy for categorizing API security risks.

This section maps the KushoAI dataset against each category, reporting both observed failure rates and the corresponding test suite coverage, defined as the share of test suites that include at least one assertion targeting that category.

The gap between these two numbers is the coverage deficit.

API1: Broken Object Level Authorization (BOLA)

Observed in dataset

22%

share of all security failures attributable to BOLA

Test suite coverage

29%

of test suites include any cross-user access probe

BOLA is the most consequential category in the OWASP list and one of the least tested. In 71% of suites that include BOLA coverage, those tests were AI-generated; manual authors almost never write cross-user access assertions unless explicitly asked.

The reason is intuitive: developers write tests from their own perspective, verifying that their user can access their resource. The adversarial question of whether a user can access someone else's resource requires a deliberate shift in mindset that most test authors don't make without a prompt.

API2: Broken Authentication

Observed in dataset

67%

of auth failures involve an edge case, not a primary flow

Test suite coverage

18%

of suites test expired token behaviour specifically

67% of authentication failures in the dataset involve an edge case rather than the primary auth flow. The most common pattern: an expired token that keeps returning valid responses because the server checks the token's structure and signature, but not its expiry timestamp.

Despite this being the most frequent auth failure type, **only 18% of test suites** include a test that sends an expired token. This is one of the widest coverage gaps in the dataset: a failure mode observed in the majority of cases, tested in fewer than one in five suites.

In practice, it means that credentials revoked after an employee departure, a partner off boarding, or a suspected breach may continue granting API access for the remainder of their token lifetime.

API3: Broken Object Property Level Authorization

Schema validation coverage

41%

of suites validate response body against a declared schema

Mass assignment coverage

14%

include any mass assignment probe

Object property level authorization, which covers both excessive data exposure and mass assignment vulnerabilities, shows a meaningful gap between schema validation and targeted security probing.

While 41% of suites validate response bodies against a declared schema, only 14% include a mass assignment probe: a request that attempts to write to a field that should be read-only or inaccessible to the requesting user.

Mass assignment failures are both common and easy to miss in manual testing because they require deliberate adversarial thinking about the request body.

API4: Unrestricted Resource Consumption

Observed in dataset

23%

of organizations show rate limit bypass as an observable failure

Test suite coverage

19%

of test suites include a rate limit probe

Rate limiting is the one category in this dataset where coverage roughly tracks observed failure rate: approximately 23% of organizations show a rate limit bypass as a measurable failure, and approximately 19% of test suites include a rate limit probe.

The remaining gap is partially explained by the difficulty of testing rate limiting deterministically in a CI/CD context: a reliable rate limit test requires precise control over request timing and volume, which standard test frameworks do not easily provide. For pricing, inventory, and data-export endpoints, that untested gap is the direct attack surface for competitive intelligence scraping and automated order manipulation.

API5: Broken Function Level Authorization

Observed in dataset

28%

of failures involving admin endpoints are accessible with standard user tokens

Test suite coverage

17%

of test suites include privilege escalation checks

Function level authorization failures, where a standard user token is accepted by an endpoint that should require elevated permissions, appear in 28% of cases involving administrative or elevated-privilege endpoints. Only 17% of test suites include any privilege escalation check (across all suites in the dataset, not filtered to those with admin endpoints).

This category is one where AI-generated tests show the largest improvement over manual suites, because generating a test that deliberately uses an under-privileged token against a privileged endpoint requires no domain knowledge; it requires only the systematic application of a security testing pattern.

API6: Unrestricted Access to Sensitive Business Flows

Multi-step flow coverage

22%

of suites include any multi-step flow validation

Abuse scenario coverage

11%

include business logic abuse scenarios

Business flow abuse, including coupon stacking, inventory manipulation, order replay, and price tampering, is the hardest OWASP category to test systematically because it requires understanding the business semantics of an API, not just its technical schema.

Only 11% of test suites include a business logic abuse scenario, and where present these tests are almost exclusively written by humans with domain knowledge rather than generated by AI systems. This is the category where automated testing has the most ground still to cover.

API7: Server-Side Request Forgery (SSRF)

Test suite coverage

8%

of suites include any SSRF test payload

Manual coverage: ~0%

AI only

Exclusively AI-generated where present

SSRF is the lowest-coverage category for manual test suites; in the dataset, SSRF test payloads are exclusively found in AI-generated test suites. Manual authors almost never include SSRF probes in standard API test suites, likely because SSRF is more commonly associated with infrastructure security testing than API-layer concerns. Only 8% of all suites include any SSRF coverage.

The practical implication is that for the 92% of organizations with no SSRF coverage, the only path to closing that gap without significant manual effort is AI-assisted test generation. This is the starkest example in the dataset of a vulnerability class where human authorship has effectively zero coverage and AI authorship has measurable coverage.

API8: Security Misconfiguration

Observed in dataset

31%

of APIs return verbose error messages in production

Test suite coverage

27%

of suites validate error response content

Security misconfiguration, including verbose error messages, missing security headers, and permissive CORS policies, is the one category where coverage tracks observations most closely. 31% of APIs in the dataset return verbose error messages in production environments, and 27% of test suites include validation of error response content. The relative alignment suggests that error message testing is well-established enough in testing culture to be included consistently, even if not universally.

API9: Improper Inventory Management

Observed in dataset

43%

of API imports surface previously undocumented endpoints

Test suite coverage

N/A

Not directly testable from suite analysis

API inventory management, covering the risk of shadow APIs, deprecated endpoints, and undocumented routes, surfaces in 43% of API imports in the dataset, where at least one endpoint discovered during import was not present in the organization's documented API surface. This is not testable from suite analysis alone, since test suites by definition test known endpoints.

The 43% figure represents a risk signal rather than a testable coverage metric; it is a strong argument for continuous API discovery as a complement to test suite analysis. An endpoint that does not appear in any test suite is an endpoint that receives no security assertions of any kind: no auth checks, no input validation, no schema enforcement. Shadow APIs are not just an inventory problem; they are a coverage exclusion.

API10: Unsafe Consumption of APIs

Test suite coverage

24%

of suites consuming external APIs include response validation

Manual coverage: ~0%

Lowest

Lowest-tested category overall

Unsafe consumption of third-party APIs, specifically the failure to validate responses from external dependencies before passing data downstream, is the lowest-tested category overall. Only 24% of test suites that consume external APIs include response schema validation (base: suites with at least one outbound third-party call, approximately 31% of all suites in the dataset).

This coverage gap is directly relevant to supply chain risk: an API that passes unvalidated third-party data into its own response surface is vulnerable to any compromise of that third-party. See Section 6 for a detailed analysis of supply chain threats and the testing gap they represent.

Industry Security Patterns

Security failure rates and test coverage patterns vary significantly by industry vertical, driven by regulatory environment, development culture, and the risk profile of the APIs being tested. The following table and analysis identify the dominant failure patterns and most significant signals by sector.

Security Failure Categories: Share of All Security-Relevant Failures

Industry	Top Failure	Auth Failure Rate[1]	Avg OWASP Coverage	Key Signal
Fintech / BFSI	Auth / BOLA	41%	7/10 categories	Highest coverage driven by compliance
SaaS / Technology	Input validation	29%	6/10 categories	Highest AI-generated test share
Healthcare / MedTech	Excessive data exposure	33%	4/10 categories	PHI exposure risk high; coverage low
E-commerce	Rate limiting	31%	4/10 categories	Inventory and pricing endpoint abuse
Enterprise / Consulting	Security misconfiguration	27%	3/10 categories	Highest manual suite share; lowest AI adoption

[1] Auth failures as a share of all security-relevant test failures observed in that vertical, not as a share of all test failures.

Fintech and BFSI

Financial services organizations show the highest OWASP coverage in the dataset, averaging coverage across 7 of 10 categories, and the highest auth failure rate at 41%. These two facts are not contradictory: high coverage means more failures are surfaced and measured, not that fewer exist.

Regulatory frameworks including PCI-DSS and RBI guidelines have created a compliance pull toward security testing that is absent in other verticals. Organizations in this sector show 2.4x higher auth edge case coverage compared to the dataset average, driven by mandated testing requirements for token management and session lifecycle.

The implication for non-regulated industries is direct: the fintech failure rate is high because fintech test suites actually look for failures. Most other industries are not looking.

SaaS and Technology

SaaS and technology companies show the highest share of AI-generated tests in the dataset, and the data reflects the coverage advantage this confers: organizations in this vertical with the highest AI adoption rates show 47% higher OWASP category coverage than those in the same vertical relying primarily on manually authored suites.

That gap is not a tooling gap; it is an adoption gap. Input validation is the dominant failure category, driven by the diversity of API consumer types and the challenge of validating inputs across multi-tenant architectures, where a validation failure for one tenant can expose data belonging to another.

Healthcare and MedTech

Healthcare organizations present the most concerning risk profile in the dataset: a high auth failure rate (33%), a dominant failure pattern in excessive data exposure, and the second-lowest OWASP category coverage at 4 of 10. HIPAA and GDPR create strong compliance incentives around data handling practices, but neither regulation translates directly into API security test requirements. The gap between regulatory intent and engineering test coverage is wider in healthcare than in any other vertical.

PHI exposure via poorly scoped API responses is the highest-consequence failure type in this sector. For healthcare teams looking to close that gap, three test types are highest priority: (1) response body scoping assertions that verify patient-identifiable fields are not returned outside their authorized context; (2) cross-user resource access probes on any endpoint that returns records keyed to a patient or user ID (a direct BOLA test for FHIR-style resource APIs); and (3) explicit scope validation for any OAuth token that accesses clinical data, verifying that read-only tokens cannot write and that patient-level tokens cannot access population-level queries. These map directly to the HIPAA Security Rule's access control requirements and are testable today with standard API testing tooling.

E-commerce

E-commerce APIs are disproportionately exposed to rate limiting and business flow abuse; the combination of high request volumes, publicly accessible endpoints, and economically motivated adversaries creates a distinct threat model.

Rate limiting is the top failure category, driven by pricing and inventory endpoints that can be queried to extract competitive intelligence or manipulate order outcomes. Only 4 of 10 OWASP categories are covered on average, with business logic abuse scenarios notably absent despite being among the highest-consequence failure types for this vertical.

Enterprise and Consulting

Enterprise and consulting organizations show the lowest OWASP coverage in the dataset at 3 of 10 categories on average, and the lowest AI test adoption rate. Testing culture in this segment is predominantly manual: suites are authored by QA teams or consultants working from documented requirements, with security testing treated as a periodic audit concern rather than an engineering discipline embedded in CI.

The dominant failure pattern is security misconfiguration: verbose error responses, missing security headers, and permissive CORS policies that accumulate in internally-deployed APIs built on the assumption that network perimeter controls are sufficient. That assumption does not hold once APIs are exposed externally, which is the standard trajectory for enterprise software under digital transformation programs. The consequence is that the organizations with the largest API surface areas and the longest-running systems are also the ones with the least security test coverage per endpoint. They are not the organizations most likely to detect a breach early.

Coverage Gap and Auth Failures in Depth

The most important finding in this dataset is not how many security tests fail; it is how many are never written in the first place. The table below maps each security test type to its OWASP category and shows what share of test suites include at least one assertion of that type. For most categories, the number is low enough that the problem is not tests failing. It is tests not existing.

Security Test Type Coverage Across Dataset

Security Test Type	OWASP	% of Suites
Unauthenticated request to auth-required endpoint	API2	91%
Expired / revoked token behaviour	API2	18%
Cross-user resource access (BOLA probe)	API1	29%
Oversized / malformed payload handling	API3, API4	38%
Rate limit enforcement	API4	19%
Error response schema validation	API8	27%
Admin endpoint access with non-admin token	API5	17%
Security header presence (CORS, CSP, HSTS)	API8	34%
Third-party API response validation	API10	24%

Percentages reflect share of all test suites in the dataset unless otherwise noted. Suites consuming external APIs (for the API10 row) represent a subset of approximately 31% of total suites.

91%

of test suites validate that authentication is required

29%

validate that it is correctly scoped

Nearly every test suite (91%) checks that an unauthenticated request gets rejected with a 401 or 403. That is the easy part. Fewer than **29% go further** and verify that authentication is correctly scoped: that a token for User A cannot retrieve User B's data, or that a read-only token cannot perform write operations.

The auth gate is universally tested. What the gate actually enforces is not. This is not a subtle distinction. An API that correctly rejects unauthenticated requests but incorrectly accepts cross-user access requests is, from an attacker's perspective, fully accessible. The 91% coverage stat describes a test that would not catch a BOLA vulnerability under any circumstances.

AI vs Manual Coverage Comparison

2.7x

More OWASP categories covered in AI-generated suites compared to manually authored suites. The delta is consistent across all 10 categories and all industry verticals in the dataset.

Note: this comparison is based on a sample review of manually authored vs AI-generated test suites from the dataset. It reflects an observed pattern rather than a controlled experiment, and should be read as directional.

The 2.7x coverage multiplier for AI-generated suites is not driven by volume; AI suites do not simply include more tests. It is driven by pattern diversity. AI systems systematically apply security testing patterns that human authors skip: cross-user access probes, expired credential tests, privilege escalation checks.

These are not tests that require deep domain knowledge to write; they require only the consistent application of a security testing checklist. This behavior is observed consistently across all 10 OWASP categories and across all industry verticals in the dataset. It is not a benchmark result. It is what actually happens when engineering teams use AI-assisted test generation on real systems at scale.

Auth Failures in Depth

Auth failures represent 38% of all security failures in the dataset, the single largest category by a substantial margin. This section breaks down the subtype distribution, the method-level failure patterns, and the relationship between auth failures and the release cycle of the endpoints involved.

The most common auth failure subtype, incorrect scope accepted at 34%, reflects a consistent implementation gap: the authorization middleware validates that a token is present and structurally valid, but does not check whether that token's scope claims actually cover the operation being performed. A read-only token that can invoke a write endpoint will pass every standard authentication test. It only fails when a test deliberately uses the wrong scope. Cross-user resource access (BOLA) is tracked separately in Section 2 under API1, where it accounts for 22% of all security failures; it is not included in these auth failure subtypes to avoid double-counting.

Auth Failure Subtypes: Distribution

Subtype	Description	Share
Incorrect scope accepted	Token with insufficient scope succeeds	34%
Expired / stale token accepted	Valid token past expiry not rejected	24%
Missing auth header accepted	No Authorization header returns 200	21%
Token after logout still valid	Session token usable after logout	12%
Malformed token accepted	Structurally invalid token not rejected	9%

What These Failures Look Like in Practice

These are not theoretical failure modes. The following patterns appear repeatedly across the dataset and are representative of how auth failures manifest in real production APIs:

1. The unguarded new endpoint
A team adds a POST /api/v1/admin/users/bulk-delete endpoint during a sprint. The route is registered directly on the base router rather than the auth-protected sub-router used by every other admin endpoint. JWT validation runs correctly on all existing routes. This one returns 200 with no Authorization header present. It passes all functional tests (no functional test checks what happens without auth) and reaches production. Detected three weeks later during a security review.

2. Cross-user data readable by any authenticated user
A GET /api/v1/reports/{report_id} endpoint validates that the request carries a valid JWT and returns 401 if not. But the authorization check stops there; it does not verify that the report_id in the path belongs to the authenticated user. Any valid token can retrieve any report by iterating IDs. The API is "authenticated" in the conventional sense; it is not authorized in any meaningful sense. This is the most common BOLA pattern in the dataset.

3. Scope not enforced on write operations
An OAuth flow issues tokens with either read or read:write scope. The backend validates token structure and signature on every request but never checks the scope claim against the operation being performed. A read-only token issued to an integration partner successfully calls PATCH /api/v1/users/{id}/profile and mutates user data. The failure is invisible to any test that uses a valid token; it only surfaces when a test deliberately uses a token with insufficient scope for the operation.

4. Token still valid after logout
A user logs out through the application. The frontend clears the stored token. The backend marks the session as inactive in its own session table. But the JWT validation middleware is stateless; it checks the token's signature and expiry timestamp only, both of which are embedded in the token itself. It never queries the session table. So from the middleware's perspective, the token is still valid for the remainder of its 24-hour lifetime, even after logout. This is not a rare implementation oversight; it is the default behavior of most stateless JWT middleware unless revocation is explicitly implemented. A test that logs out and immediately retries a request with the original token would catch this. Fewer than 12% of test suites include that sequence.

Auth Failures and the Release Cycle

3.1x

higher auth failure rate on endpoints in their first 30 days

New endpoints carry a disproportionate share of auth failures. Endpoints in their first 30 days of production availability have a **3.1x higher auth failure rate** than endpoints older than 90 days, and this pattern holds consistently across all verticals and endpoint types. The explanation is straightforward: new endpoints get added in feature branches under time pressure, with authorization logic copied from nearby endpoints that may not correctly inherit the scope requirements for the new functionality, and with the least test coverage. Security testing should be most rigorous for the newest code. The data shows the opposite is true.

This finding is significant beyond the auth failure rate itself. The ability to observe failure rates by endpoint age is a function of platform-level telemetry across thousands of organizations, not something visible from inside a single team's test suite. The practical implication is that security coverage should be weighted toward recently added endpoints in every CI pipeline. The organizational implication is that the riskiest moment in any API's lifecycle is the period immediately after it ships.

What to Do Now: The Five Highest-Impact Security Tests Missing From Most Pipelines

The data in this report points consistently to a coverage gap, not a capability gap. The failures documented here are not novel attack techniques requiring specialized tooling; they are standard vulnerability classes that automated tests would surface, running in pipelines that already exist. These are the five test types with the highest combined impact relative to their implementation cost, ranked by the failure rate and coverage deficit data in this dataset. Each represents a category where the risk is high, the test is straightforward, and the majority of organizations are currently running blind.

1. Cross-user resource access probe (BOLA)

Business risk: Any authenticated user can access any other user's data by iterating resource IDs. This is the most commonly exploited API vulnerability class in breach disclosures and accounts for 22% of all security failures in this dataset.
For every endpoint that returns a resource identified by an ID in the path or query string, add a test that requests that resource using a valid token belonging to a different user. The expected response is 403. If it returns 200, you have a BOLA vulnerability. This test is absent from 71% of test suites in this dataset.

2. Expired token test on every auth-required route

Business risk: Credentials revoked after an employee departure, a partner offboarding, or a suspected breach may continue granting API access for the full token lifetime. For a 24-hour JWT, that window is 24 hours of continued access after revocation.
Generate a token, wait for it to expire (or backdate the expiry claim in a test environment), and verify that the API returns 401. This catches the most common auth edge case in the dataset: expired tokens accepted because middleware checks structure, not expiry. Only 18% of suites currently include this test.

3. Underprivileged token against every elevated-permission endpoint

Business risk: Standard user accounts can invoke administrative functions, including bulk operations, user management, and data deletion endpoints. This pattern appears in 28% of cases involving admin endpoints in this dataset.
For any endpoint that should require admin or elevated scope, add a test using a standard user token. The expected response is 403. This catches function-level authorization failures, present in 28% of cases involving admin endpoints and tested in only 17% of suites.

4. No-auth request against every write endpoint

Business risk: Write endpoints added during active development frequently miss auth middleware entirely, reaching production with no authentication enforcement. DELETE endpoints show the highest auth failure rate in this dataset at rates significantly above GET equivalents.
For every POST, PUT, PATCH, and DELETE endpoint, add a test with no Authorization header. Expect a 401. This catches routes added to the wrong router group that never get the middleware applied. One test per endpoint catches the most common and most consequential class of auth failure.

5. Response schema validation on every external API call

Business risk: An API that passes unvalidated third-party data into its own response surface inherits any compromise of that third-party. As supply chain attacks targeting AI APIs and LLM integration libraries increase in frequency, unvalidated external responses are a direct injection path into production systems.
For any endpoint that calls a third-party API and passes data from that response into its own response, add an assertion validating the third-party response schema before it is used. This is the minimum testable defense against supply chain data injection and is currently present in only 24% of suites that consume external APIs.

Supply Chain Attacks: The Untested Frontier

Supply chain attacks target the infrastructure around your API, not the API itself. They compromise the packages your API depends on, the build pipeline that produces it, or the third-party services it consumes. The OWASP API Top 10 framework, and every dynamic API security testing tool built around it, is designed to test the behavior of APIs you own and control. That scope is becoming structurally insufficient.

The incidents documented in this section are not edge cases. They represent a consistent and accelerating pattern: attackers have moved up the stack, from exploiting vulnerable API endpoints to compromising the build infrastructure, package registries, and AI integration libraries that produce and run those endpoints. The KushoAI platform data has one direct signal here: only 24% of test suites that consume external APIs include response schema validation before passing that data downstream, the lowest-tested category in the entire dataset. Everything beyond that data point is drawn from public incident reports, CISA advisories, and security community disclosures. The picture they form is the same: the boundary of what constitutes API security has expanded, and the testing toolchain has not kept up.

A Rapidly Expanding Attack Surface

The scale of the supply chain threat has grown by orders of magnitude over five years. The escalation is not gradual; it is structural, driven by the expansion of the open source ecosystem, the proliferation of automated CI pipelines that execute third-party code directly, and the increasing value of the credentials and API keys that live inside those pipelines.

2021: Baseline

Approximately 650 documented software supply chain attacks (Sonatype State of the Software Supply Chain). Package registry poisoning is an observed but niche attack class. Most security teams treat it as a background risk rather than a primary threat vector.

2022–2023: Volume surge

88,000+ malicious packages detected across npm, PyPI, and RubyGems in 2022, a 135x increase over the 2019 baseline. By 2023 the count reaches 245,000+. Package registry poisoning is no longer niche; it is a systematic, scaled attack category. CISA reports a 2× increase in supply chain incidents affecting critical infrastructure.

2024: Sophistication escalates

512,000+ malicious packages flagged across major registries. The XZ Utils backdoor reveals nation-state-level patience and technical sophistication applied to open source infrastructure. LottieFiles demonstrates that a single compromised maintainer token is enough to weaponize a widely-used UI library against hundreds of thousands of end users.

2025–2026: AI APIs become a target

Supply chain attacks begin targeting AI-specific infrastructure: the credentials, API keys, and provider tokens managed by LLM integration libraries. The LiteLLM PyPI attack in March 2026 and the Shai-Hulud npm worm campaigns documented by Palo Alto Networks Unit 42 demonstrate that attackers are actively adapting to the expanding credential surface created by AI-native backends. The attack surface has shifted: the target is no longer just the API, it is the AI infrastructure running beneath it.

Notable Incidents (2024–2026)

XZ Utils Backdoor, March 2024

A sophisticated multi-year social engineering campaign introduced a backdoor into the XZ Utils data compression library, specifically targeting OpenSSH on systemd-based Linux distributions. The campaign involved a fictitious contributor identity built over two years of legitimate contributions before the malicious commit was introduced. It was caught only by a Microsoft engineer who noticed anomalous CPU usage in an unrelated benchmark. Had it reached major production distributions, it would have compromised SSH authentication across a significant fraction of internet-connected Linux servers, triggered without any API request and invisible to any API security test.

LottieFiles npm Compromise, October 2024

The @lottiefiles/lottie-player npm package was compromised via a stolen maintainer token. Malicious versions (2.0.5–2.0.7) injected a cryptocurrency drainer into any site that loaded the package from a CDN or npm directly. Losses exceeded $700K before the malicious versions were pulled. Applications consuming the library behaved correctly on all their own endpoints; the compromise was entirely within the dependency, undetectable by any API-layer test targeting the consuming application.

tj-actions/changed-files, March 2025 (CVE-2025-30066)

A widely used GitHub Actions CI action (tj-actions/changed-files, used by 23,000+ repositories) was compromised to dump repository secrets into workflow logs. Any repository running the action during the attack window had its CI secrets (API keys, cloud credentials, signing tokens) exposed in public or internal logs. The attack vector was the CI pipeline itself, not the application being built. Standard API security testing cannot observe what happens inside a build pipeline or what secrets are accessible to CI actions.

Shai-Hulud npm Worm, November 2025

A self-propagating npm package worm affected 500+ packages including dependencies used by Zapier, PostHog, and Postman. The worm targeted packages with publish permissions by compromising maintainer sessions and publishing malicious patch versions that added the worm's own payload. The infection chain was entirely within the package registry; no application API was involved at any stage. First-party API security testing had no visibility into the attack surface.

LiteLLM PyPI Supply Chain Attack, March 2026

Attackers published a set of malicious PyPI packages with names closely mimicking LiteLLM plugins (e.g., litellm-openai-proxy, litellm-anthropic-plugin). The packages exfiltrated AI API keys (OpenAI, Anthropic, Cohere) from any environment that imported them, targeting the growing number of AI-native backends that manage large numbers of provider credentials. The attack is notable for specifically targeting the AI API layer: as LLM-integrated APIs proliferate, the credential surface they manage becomes a high-value target for supply chain attacks engineered to reach that layer.

The Testing Gap

The current API security testing toolchain, including the OWASP API Top 10 framework, dynamic analysis tools, and AI-generated test suites, is designed to test the behavior of APIs you own and control. It tests whether your endpoints correctly enforce authentication, reject invalid inputs, and return appropriate status codes. It does not, and structurally cannot, test:

1. Dependency Integrity
Whether the third-party libraries your API runtime depends on have been tampered with at the package registry level. A malicious package that passes all functionality tests will also pass all security tests, because it behaves correctly until it doesn't.

2. Build Pipeline Security
Whether your CI/CD artifacts were produced from the source code you reviewed and committed. A compromised build environment can inject malicious behavior after the source-level security review and before the artifact reaches production.

3. CDN and Upstream Asset Integrity

Whether third-party scripts and assets your application loads at runtime have been modified. CDN-layer compromise is invisible to any test that does not independently verify the hash of every loaded asset against a known-good baseline.

4. Transitive Dependency Exposure

Whether a dependency two or three levels deep in your dependency tree carries a known or zero-day vulnerability. The average production API depends on dozens of direct packages and hundreds of transitive ones, each a potential attack surface that no API security test currently addresses.

In the KushoAI dataset, **only 24%** of test suites that consume third-party APIs validate the response schema before passing that data downstream, the most basic check for OWASP API10. The deeper supply chain risk, covering compromised packages, poisoned pipelines, and compromised CI actions, has **no corresponding test category in any current automated testing framework**. There is no test you can write today that will tell you whether a package you installed last week has been tampered with since.

That gap is not a criticism of the OWASP framework or of the tools built around it. It is a structural consequence of how API security testing was defined before supply chain attacks became a primary threat vector. The framework tests what your API does. It was never designed to test what your API is built from.

The trajectory documented in this section points to a necessary expansion of scope. First-party API security testing needs to be complemented by supply chain signal: continuous dependency integrity monitoring, build artifact verification, and response schema validation for all third-party API consumption treated as a mandatory assertion rather than an optional check. These are not speculative capabilities. The tooling exists in adjacent spaces: SCA tools, SBOM platforms, CI security scanners. But it is not integrated into the API testing workflow where the coverage gap actually lives.

For engineering organizations, the immediate question is not whether supply chain attacks will affect their API infrastructure. The incidents documented here span early-stage SaaS products and large financial institutions, automated package managers and manually pinned dependencies, internal-only APIs and publicly exposed ones. The question is whether the testing coverage in place today would surface a compromise before it reaches production. For most organizations, based on the data in this report, the answer is no.

AI's Role in Security Testing

AI does not make APIs more secure. It surfaces failures that were already there. Every security vulnerability found by an AI-generated test suite existed before the test ran; the AI did not create the vulnerability, it just looked for it more systematically than a human author would. That is the actual value: not novel attack generation, but consistent application of a known security testing checklist across all of the API surface, every time.

The coverage data is unambiguous on this. AI-generated suites cover more OWASP categories, produce higher coverage rates within each category, and generate fewer false positives when reviewed by a human. The reason is not that AI is smarter about security; it is that AI applies a security testing checklist consistently, without the blind spots and shortcuts that human authors naturally develop. The 2.7x coverage figure reflects observed behavior across organizations in this dataset, not a controlled benchmark. It is consistent across all 10 OWASP categories and across all industry verticals.

Where AI Outperforms Manual Authorship

Auth edge cases. AI-generated suites cover auth edge cases at 78% vs 31% for manual suites. Almost all of that gap is in the edge cases: AI systems generate tests for expired tokens, revoked credentials, tokens used after logout, and malformed authorization headers as a matter of course. Manual authors write the happy path reliably and edge cases only when they remember to.

BOLA probes. The 46 percentage point gap in BOLA coverage is the second largest in the dataset. Testing for BOLA requires asking an adversarial question: can my token access a resource that belongs to a different user? That question does not come naturally to a developer writing tests for their own feature. AI systems ask it automatically.

Boundary inputs across all fields. AI-generated suites send oversized payloads, malformed values, and out-of-range numbers across the entire API surface, not just the fields that look obviously security-sensitive. Manual authors tend to focus boundary testing on passwords, tokens, and IDs, and skip fields that seem low-risk. Production vulnerabilities do not respect that distinction.

Suite Type Comparison: Coverage, Edge Cases, and Quality

Suite Type	Avg OWASP Coverage	Auth Edge Case Coverage	False Positive Rate
Manually authored	26%	31%	12%
AI-generated, no human edit	71%	78%	9%
AI-generated, human-edited	84%	91%	4%

Human-edited AI suites outperform unedited AI suites on all three dimensions. The false positive rate drops from 9% to 4% when a human reviews the AI output, removing tests that are structurally correct but contextually wrong for that specific API. Coverage climbs from 71% to 84% as human judgment fills in domain-specific flows that pattern-based generation misses.

This pattern is consistent across every vertical and organization size in the dataset. The workflow that produces the best security coverage is not a choice between AI and human authorship. It is AI generating the baseline at scale, applying the security checklist consistently across all surface area, and a human refining it with the business context and domain knowledge the model cannot derive from an API schema alone. Organizations that have adopted this workflow show the lowest security failure rates in the dataset.

Looking Ahead

Security Testing Moves from Compliance Checkbox to Continuous Assertion

The organizations in this dataset with the lowest security failure rates do not have the most thorough security audits. They have security assertions in their CI pipeline: tests that run on every commit and block deployment on failure, the same as any functional test. The shift from "security as a periodic review" to "security as a continuous assertion" is underway, but unevenly. The tooling exists. The gap is adoption.

BOLA and Auth Scope Validation Become Table Stakes in CI Pipelines

The highest-impact near-term change is not sophisticated attack simulation. It is adding two specific test types to every CI pipeline: cross-user access probes (BOLA) and token scope validation tests. Both are structurally simple, both can be generated by AI without domain knowledge, and together they address the two most common failure categories in this dataset. Teams that add just these two test types to their standard CI suite will close the majority of their auth and authorization coverage gap.

The Gap Between Security Team Findings and Engineering Test Coverage Will Close

Security teams find failures that engineering test suites miss. Part of this is mindset: security teams think adversarially, engineers think functionally. Part of it is tooling: security testing runs in a separate workflow, disconnected from the CI pipeline. AI-assisted test generation at the engineering layer narrows this gap by applying adversarial patterns at the point where tests are written. The end state is not two parallel testing disciplines; it is one suite that covers both functional correctness and security assertions, running in the same pipeline, enforced at the same deployment gate.

The shift from periodic security review to continuous security assertion is not a gradual evolution. It is a response to a structural change in how software is built and attacked. Release cycles have compressed from quarterly to daily. Attack tooling has been automated.

The compliance frameworks that drove security investment in financial services and healthcare are extending into adjacent verticals. These forces converge on a single outcome: security testing embedded in the engineering workflow, at the cadence of the engineering workflow, is becoming a baseline expectation rather than a competitive differentiator.

The data in this report represents an early cross-section of that transition, observed from inside the testing activity of the organizations navigating it.

Conclusion

The data in this report points to a consistent pattern across 1.4 million test executions and 2,600+ organizations: API security failures are not primarily caused by sophisticated attack techniques or novel vulnerability classes. They are caused by the systematic absence of edge case testing for authentication and authorization logic that is already present in the codebase.

The tools to close this gap are available now. AI-assisted test generation covers 2.7x more OWASP categories than manually authored suites. Human review of AI-generated suites reduces false positive rates by more than 50% and increases coverage further. The organizations in this dataset that demonstrate the lowest security failure rates have made a single structural change: they test what happens when authentication is not just missing, but wrong: insufficient scope, expired credentials, cross-user access, post-logout validity. These are not exotic tests. They are the tests that most suites do not yet include.

Supply chain security represents the next testing frontier, one where the current toolchain has no coverage at all. As the incidents documented in Section 6 demonstrate, the attack surface has expanded well beyond the API endpoints that security testing frameworks currently address; it is now actively targeting the AI API layer. Closing that gap will require both new tooling and a broader definition of what API security testing means. KushoAI's position at the intersection of AI-native test generation and API security coverage puts it at the center of both the problem this report documents and the direction the market is moving.