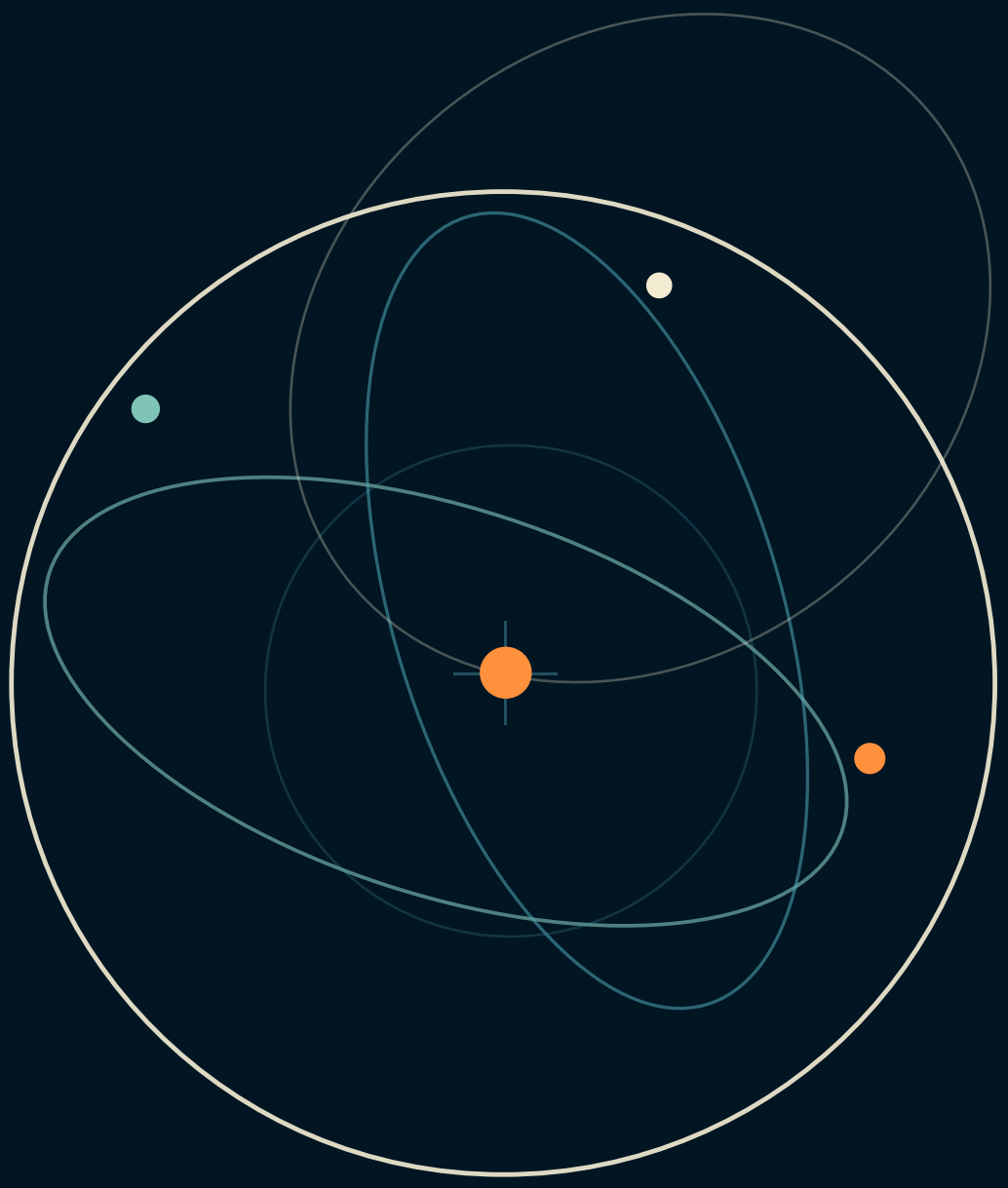
CANTINA

Apex Field Report

Autonomous Adversarial Discovery of Web-Application Vulnerabilities at Scale

An Empirical Study of 4,032 Machine-Generated Findings Across 612 Production Targets

July 2026 Pranamya Keshkamat



All target identities, endpoints, and reproduction details are redacted or paraphrased. This document reports aggregate statistics and de-identified case studies only.

Abstract

Many teams can now write software faster than they can review it, and the security defects that reach production are changing with it. Apex is an autonomous offensive-security system designed for this environment. It examines live web applications and APIs, develops attack hypotheses, and looks for evidence that a suspected weakness can be exploited. This paper summarizes its first quarter of operation. From 8 April through 17 July 2026, Apex produced 4,032 CVSS-scored findings across 612 production targets at software, financial-technology, and enterprise organizations. On the system's own scores, 702 findings were high severity or above and 124 were critical. At least one high-or-critical finding appeared on 37.4% of the targets. Nearly every finding was network-reachable (99.9%), and 93.4% required no user interaction. The most common classes were exposed secrets and data, broken authentication, and broken access control; a smaller group involved injection and remote code execution. Six redacted case studies show what those categories looked like in practice, including cross-account token theft, account takeover, and server-side template injection. Within this target set, the results show how often basic security boundaries can be missed when software changes faster than teams can examine it.

Disclaimer

This paper is published for general informational purposes only and describes, at a technical level, aggregate statistical observations and de-identified case studies drawn from the operation of the Apex system. It does not constitute, and must not be relied upon as, security, legal, or professional advice, nor as a representation about the security posture of any specific organization, product, or codebase.

All case studies are redacted. No target name, domain, endpoint, credential, reproduction step, or reporter identity is disclosed; sectors are generalized and technical particulars are paraphrased to the minimum needed to convey the class and impact of a defect. Where a finding is described as triaged or resolved, that reflects the disposition assigned by the affected vendor's own security team.

Vulnerability-class labels are produced by automated classification of finding text. Each finding is assigned a single primary class, and the labels are descriptive rather than authoritative. CVSS vectors are generated by the system and represent its assessment, not an external standard body's.

The methodology of the Apex system is proprietary and is described here only to the extent required to interpret the reported statistics. Forward-looking statements and interpretations, including those concerning AI-assisted software development, reflect the author's analysis as of the date of publication and are subject to change. To the maximum extent permitted by law, the author and Cantina disclaim all liability arising from any reliance on this paper.

1. Introduction

Software teams can now produce and ship code much faster than they could a few years ago. Language models account for some of that increase, but the security problem is broader than who or what wrote a given line. A feature can work exactly as intended and still omit an authorization check, cross a tenant boundary, expose a credential, or accept input that should have been rejected. Those controls often depend on context outside the file or prompt that produced the feature.

Reviewing that context takes time. A periodic human review cannot cover every change on every live service, especially when several teams are shipping in parallel. Apex was built to add continuous adversarial testing to that process. A fleet of autonomous agents maps a target's attack surface, tries to demonstrate concrete abuse, and then subjects candidate findings to a separate assessment and adjudication process. The system can work across hundreds of targets in parallel.

This paper focuses on the resulting findings rather than the proprietary mechanics of the system. Section 2 explains enough of the workflow to interpret the data. Sections 3 through 6 describe the corpus, severity, reachability, and major vulnerability classes. Section 7 presents six redacted cases that affected vendors triaged or resolved, except where noted. The final sections discuss where risk concentrated, how the results relate to AI-assisted development, and the limits of the study.

2. The Apex system

Four parts of the workflow matter when reading the results. More detailed information about target selection, models, prompts, and adjudication remains proprietary.

How the system hunts. Apex is not a signature scanner. Its agents map the exposed surface of a target, form hypotheses about possible abuse, and try to show impact. A candidate moves forward only when the system can describe a concrete abuse path and, where feasible, exercise it.

How a candidate becomes a finding. The initial search is intentionally broad. Candidates then pass through an assessment stage, which assigns a CVSS v3.1 vector, and a separate adjudication stage. The 4,032 findings in this paper cleared that internal reporting threshold. This does not mean that an outside party confirmed all 4,032. External disposition is reported only for the six cases in Section 7, and only where the affected vendor provided it.

How the system scales. Each target is analyzed in an isolated environment, allowing many targets to be tested at once. That parallelism is what made a corpus of this size possible in one quarter. The monthly totals in Section 4 mainly reflect how much the system was run in each period, and should not be read as a growth trend.

How severity is reported. Every finding carries a machine-generated CVSS vector. This paper uses the corresponding score bands throughout: low (< 4.0), medium (4.0-6.9), high (7.0-8.9), and critical (at least 9.0). Using one scale keeps comparisons within the corpus consistent, while the limitations of machine-assigned scores are discussed in Section 10.

3. Study design and corpus

Scope. The dataset contains findings produced by Apex against its authorized web and API bug-bounty targets from 8 April through 17 July 2026. It does not include smart-contract audits, internal model evaluations, or benchmark runs. Those activities use different systems and corpora and are not comparable with live web-application testing.

Population. During the study window, Apex produced 4,032 CVSS-scored findings across 612 production targets. Of these, 702 (17.4%) scored as high severity or above (CVSS 7 and above), including 124 (3.1%) that scored as critical (CVSS 9 and above). Figure 1 shows those tiers. Unless noted otherwise, the distributions that follow use all 4,032 findings.

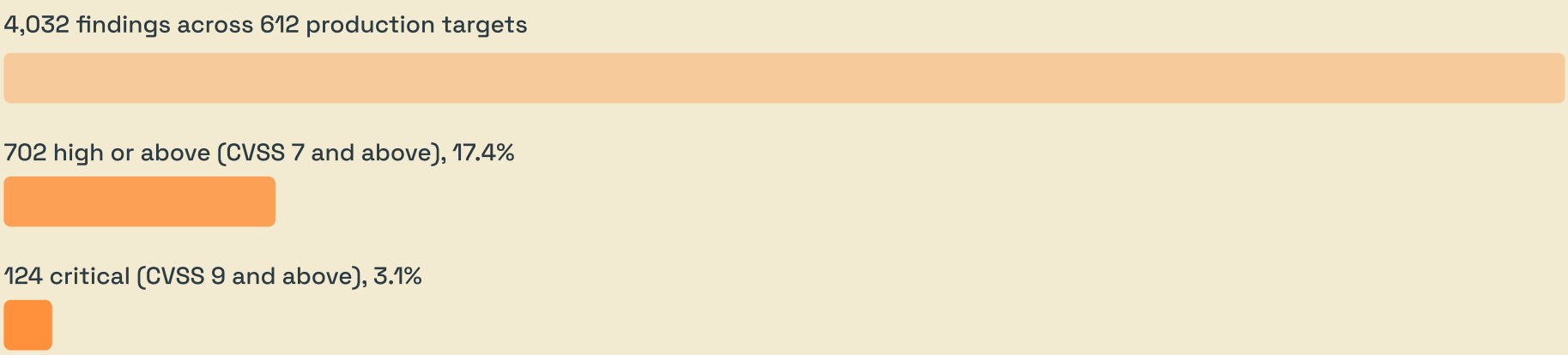


Figure 1. Findings by severity tier. One finding in six reaches high severity or above, and roughly one in thirty-three is critical. These are the system's own assessments, not external confirmations.

4. Severity and volume

4.1 Severity distribution

Most findings were medium severity (81.3%), while low-severity findings made up only 1.3%. The high-impact group was smaller but substantial in absolute terms: 578 high and 124 critical. Those 702 high-or-critical findings are what a team most needs to investigate, even though medium-severity issues dominate the corpus.

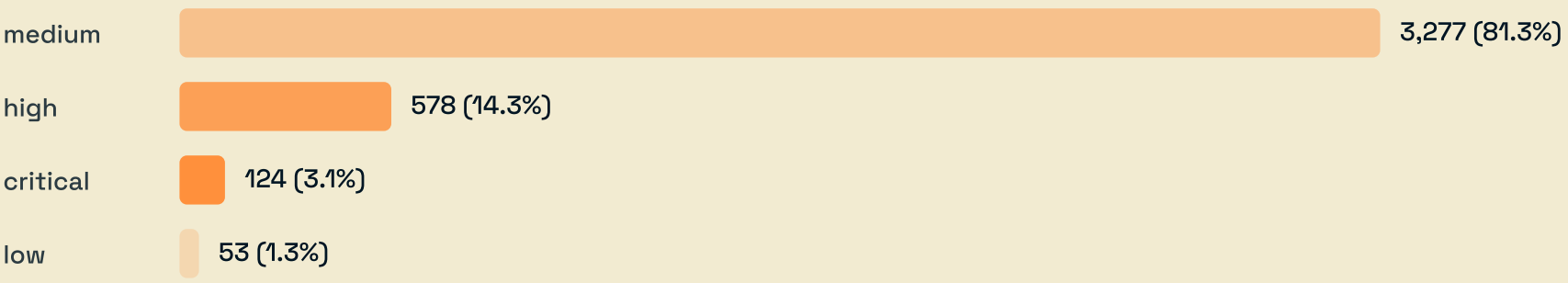


Figure 2. Severity distribution of 4,032 findings by CVSS band. The high-and-above tail is 702 findings (17.4%); the mean CVSS is 5.79.

4.2 Distribution over the quarter

Figure 3 breaks findings out by calendar month. Both ends of the window are partial (the run began 8 April and closed 17 July), so April and July are not comparable to the full months between them. Volume was uneven; it is shown for completeness, not as a trend.

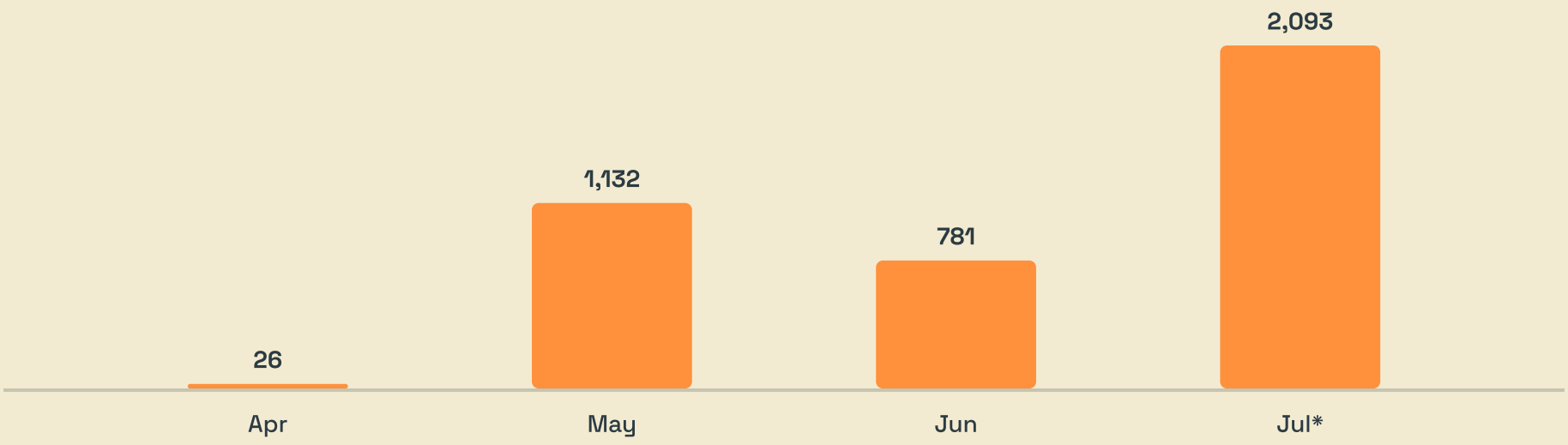


Figure 3. Findings by calendar month. April and July are partial months. Critical findings per month were 1, 46, 5, and 72 respectively. The breakdown reflects how much the system was run, not a growth trend.

5. Exploitability profile

CVSS severity describes impact together with exploitability. Three of its base metrics are especially useful here. As Figure 4 shows, Attack Vector was Network for 99.9% of findings, User Interaction was None for 93.4%, and Privileges Required was None for 53.0%. In practical terms, remote and non-interactive weaknesses were common, and just over half of the findings did not require an authenticated position. These are marginal distributions, so they do not establish the exact combination of metrics for every finding.

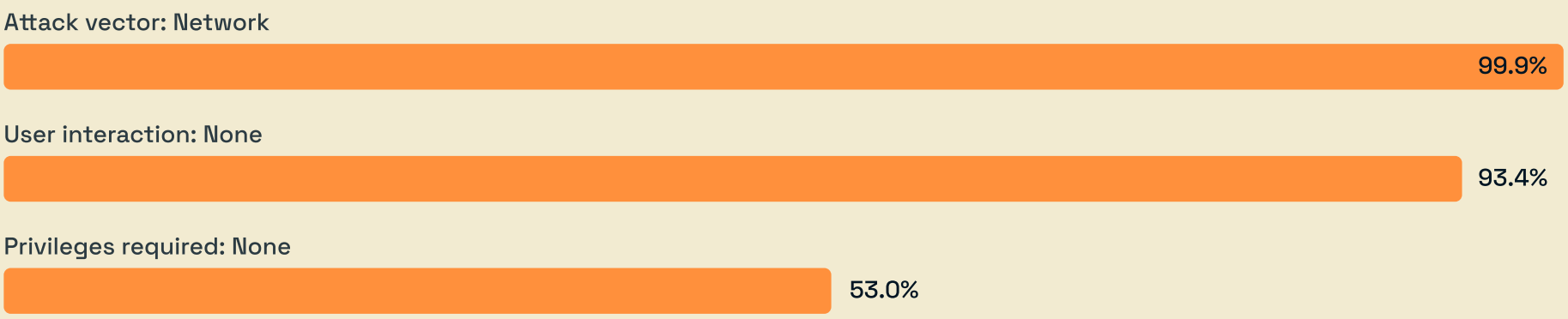


Figure 4. Exploitability profile over the 4,032 findings, each bar the share at the attacker-favorable extreme of one CVSS base metric.

This profile is consistent with weaknesses in controls at exposed boundaries: an endpoint without an authentication gate, a resource returned without an ownership check, or a secret included in a client bundle. Reachability metrics alone cannot show the root cause of a finding, however. The class distribution in Section 6 provides the stronger evidence for that interpretation.

6. A taxonomy of what breaks

Figure 5 assigns each finding to a single primary class, so the counts do not overlap. Roughly 18% of findings did not map cleanly to one class and are omitted from the chart. Bar length and orange intensity show finding volume, with stronger tones for larger categories and lighter tones for smaller ones. The largest classes generally reflect a missing control; smaller classes more often involve a control implemented incorrectly. The distinction is a useful summary, not a judgment about every finding in a category.

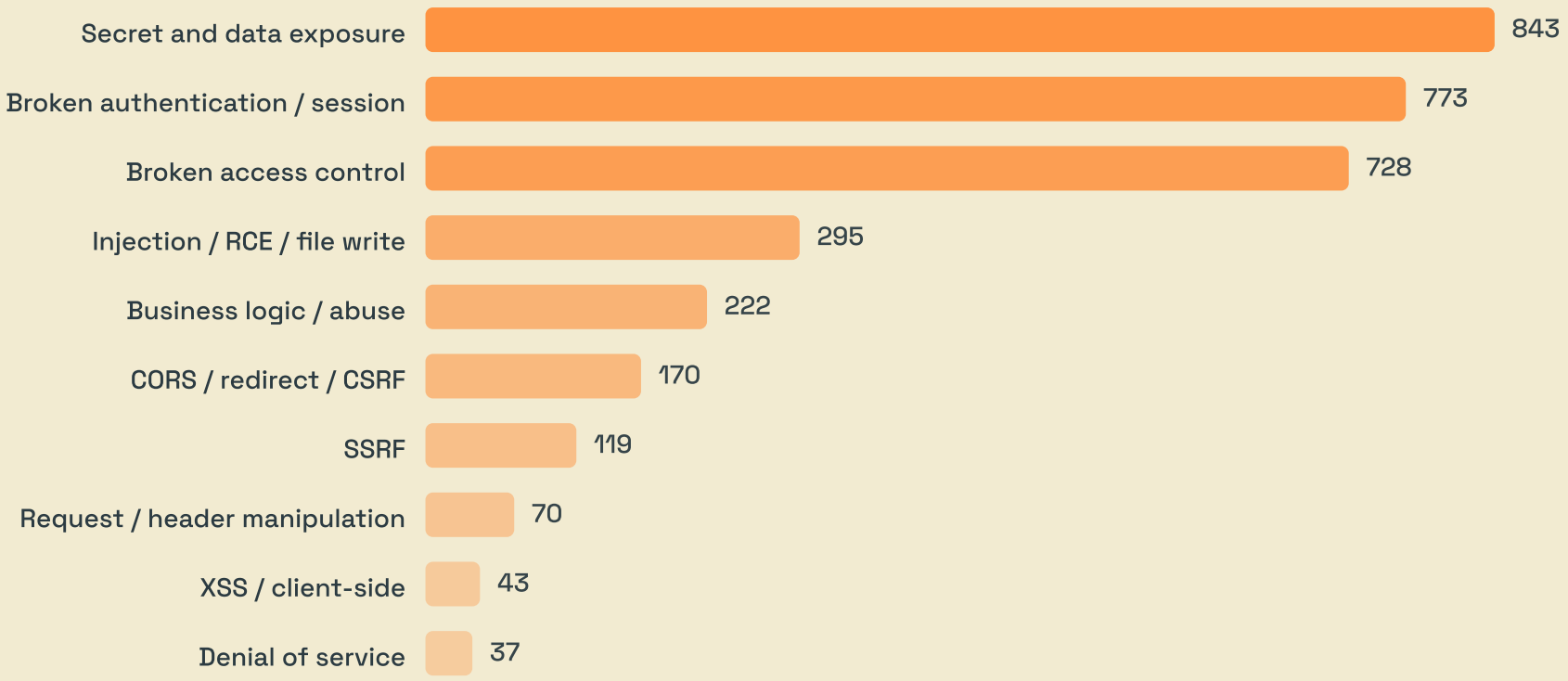


Figure 5. Vulnerability classes, one primary class per finding. About 18% did not map to a single class and are omitted. The three largest are all missing-control classes.

The three largest classes were secret and sensitive-data exposure (843 findings), broken or missing authentication (773), and broken access control (728). Together, they account for more than half of the findings classified in the chart. Each involves a security boundary that application code must preserve across many local changes. Injection, remote code execution, and arbitrary file write accounted for 295 findings. That group was smaller, but its potential impact was often greater: a template engine, deserializer, or file handler can turn untrusted input into code on the server. Section 7 includes examples. The remaining categories each represented less than 6% of the full corpus.

7. Selected findings (redacted)

Aggregate counts can make the underlying bugs feel abstract. The six cases below show what several high-impact findings looked like in practice. Each description is limited to the target's sector and a paraphrased mechanism. The findings were reported to the affected organizations and, except where noted, triaged or resolved by their security teams. That disposition applies to these cases only; it should not be generalized to the full corpus. Names, domains, endpoints, credentials, and exact reproduction steps are withheld.

A. Cross-account token theft via web-cache deception

AI writing-assistance platform, broken authentication, vendor-resolved

An authenticated API endpoint returned a live OAuth access token for the current user. With a crafted suffix on the path parameter, the edge cache stored that user-specific response under a cacheable key. When a second authenticated user requested the same URL, the cache returned the first user's token. That token then worked against the identity API as the victim. A missing cache-control rule on a credential-bearing response therefore allowed cross-account impersonation without victim interaction.

`CVSS 7.4 (High). Disposition: resolved.`

B. Victim-independent account takeover via HTTP parameter pollution

European financial-market-infrastructure platform, broken authentication, critical

A password-reset flow accepted duplicate request parameters. One layer used them to choose the account being reset, while another used them differently when deciding where to send the unlock code. By adding an attacker-controlled value, an attacker could select a victim's account but receive that victim's reset secret. The attack required neither the victim's participation nor a secret already held by the attacker.

`CVSS 9.1 (Critical).`

C. Server-side template injection to remote code execution

Enterprise reporting / export SaaS, injection/RCE, critical

A self-registered user could save an export template that the server rendered without an effective sandbox. Expressions in the template ran inside the rendering context, allowing arbitrary commands and arbitrary file reads and writes on the application server. The attack required an ordinary account, but no stolen credential or victim interaction.

`CVSS 9.9 (Critical).`

D. Mass PII exposure via a leaked live API credential

Global digital-payments provider, secret exposure, vendor-resolved

A public code-sandbox page, published as integration documentation, included a live Basic-auth credential for a payments REST API in client-side JavaScript. The credential could do far more than its documented single-endpoint purpose. It allowed paginated access to hundreds of payee records, including names, dates of birth, government identifiers, and addresses. It could also mint session tokens for arbitrary users and initiate cross-program payment operations. A public secret and missing server-side authorization combined to create the exposure.

`CVSS 9.3 (Critical). Disposition: resolved.`

E. Unauthenticated remote code execution via a document-processing sink

Global ride-hailing platform, injection/RCE, vendor-resolved

An unauthenticated document-upload path passed attacker-controlled input to a rendering component. Its interpreter sandbox could be escaped, turning an ordinary file-processing feature into unauthenticated remote code execution on the server.

`CVSS 9.8 (Critical). Disposition: resolved.`

F. Stored cross-user XSS via a trusted real-time message field

Consumer social-media platform, XSS / client-side, critical

A low-privileged user could send a crafted real-time chat event that the backend accepted as a trusted “tip” message. The server copied attacker-controlled fields into the room’s shared message stream without tying them to a real transaction and without encoding one value. The client then built markup from that value, causing stored script execution for other users who opened the room. One message could run attacker code in each viewer’s authenticated session and trigger account actions on their behalf.

`CVSS 9.6 (Critical).`

The cases cover several parts of the high-impact tail: two authentication or control omissions (A and B), two code-execution paths (C and E), one secret exposure (D), and one client-side injection (F). All were remotely reachable, and none required more than an ordinary account. They were selected to show technical range and impact, not to serve as a representative sample of a corpus in which 81.3% of findings were medium severity.

8. Where risk concentrates

8.1 Concentration

Exposure is not uniform across targets. Among the 612 production targets, the median carried 4 findings and the most affected carried 55, so a small number of targets contributed far more than the median. More importantly, 37.4% of targets contained at least one high-or-critical (CVSS 7 and above) finding (Figure 6), meaning Apex found at least one high-severity issue on more than a third of the applications it analyzed.

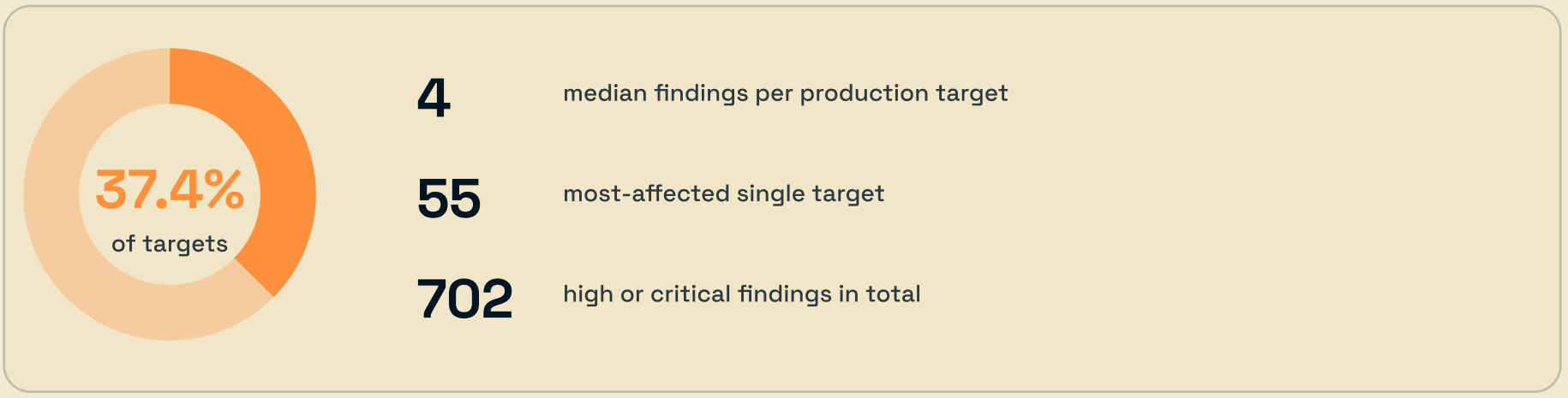


Figure 6. High-severity concentration. More than a third of targets held at least one high or critical finding; per-target counts ranged from a median of 4 to a maximum of 55.

8.2 Which organizations

The target set is drawn from public and private bug-bounty programs and skews toward organizations that already invest in security review. Sectorally it concentrates in financial technology and payments, developer and infrastructure tooling, enterprise identity and SaaS, and consumer application platforms. These were active production systems at organizations that invite outside security research, not a random sample of all web applications, so the results should not be treated as a prevalence estimate for the internet as a whole.

8.3 What this says about AI-assisted development

The corpus does not record whether a vulnerable line was written by a person or a model. It cannot establish that AI-assisted development caused any individual finding, nor does it provide a pre-AI baseline. What it does show is a pattern that matters when teams use AI to increase development speed:

1. **Composition. The leading vulnerability classes are all missing-control classes: secret exposure, broken authentication, and broken access control (Section 6).**
2. **Reachability. Network attack vectors (99.9%) and no required user interaction (93.4%) were common, while 53.0% of findings required no privileges (Section 5).**
3. **Prevalence. More than a third of the production targets had at least one high-or-critical finding (Section 8).**

These are familiar software-security failures, whether the code is handwritten or generated. AI assistance can make them harder to manage: it increases the amount of code a team produces without increasing the time available for threat modeling and review. The practical risk comes from the gap between the rate of change and the capacity to review it, regardless of who authored the code.

9. Discussion

Happy-path tests miss many of these defects. A missing authorization check can pass a test that authenticates as the intended user. An exposed secret can pass when no test inspects the built artifact. A broken tenant boundary will not show up in a single-tenant fixture, and inconsistent parameter handling may look fine for every well-formed request. Functional tests remain necessary, but these examples need negative, cross-tenant, and adversarial cases as well.

Security review has to keep up with delivery. Human judgment remains essential, including in the vendor triage behind the case studies. But a periodic, serial review process has limited coverage when code changes continuously across many services. Automated adversarial testing can broaden that coverage and give reviewers concrete leads. Its usefulness depends on disciplined adjudication: a system that produces more candidates without filtering them simply moves the bottleneck to the people who must investigate them.

Automation changes the economics for both sides. Parallel agents can test more hypotheses than a small team could examine manually, but attackers can use the same leverage. Defenders gain the most when they run this kind of testing against their own systems, connect it to remediation, and do so before the same attack paths are found elsewhere.

10. Reading these numbers

Several limits affect how these numbers should be read. First, bug-bounty targets are self-selecting. They may have stronger security programs than average, but they may also expose unusually broad or complex attack surfaces. The sample therefore cannot support claims about vulnerability prevalence in software generally. Second, severity, reachability, and class labels come from Apex's own assessment pipeline. The paper does not estimate false-positive rates, compare the system with a control method, or report independent review of every finding. Vendor triage or resolution of the six cases in Section 7 supports those examples, not every label or CVSS score in the corpus. Third, each finding receives one primary class even when several categories could apply, and about 18% did not map cleanly to the displayed taxonomy. Finally, the month-to-month variation in Figure 3 reflects how much the system was run in each period, not a change in the amount of vulnerable software, and both end months are partial.

11. Conclusion

Over one quarter, Apex produced 4,032 findings across 612 production targets. Most were medium severity, but 702 scored as high or critical, and 37.4% of targets had at least one finding in those bands. Nearly all findings were network-reachable, and the largest categories involved secrets and data exposure, authentication, and access control. The redacted cases show the possible impact, from cross-account token theft to unauthenticated remote code execution.

The results highlight a practical problem for teams adopting faster development tools: review capacity does not grow automatically with code output. Continuous adversarial testing can help close that gap when it produces reproducible evidence, filters its own results carefully, and remains connected to human triage and remediation. That is the role Apex is intended to play.