Cantina

FIELD REPORT

Apex Black

Autonomous Adversarial Discovery of Web-Application Vulnerabilities at Scale

What we observed across 612 production targets in 100 days of continuous adversarial testing, 8 April to 17 July 2026. 4,032 machine-reported findings.

July 2026 Pranamya Keshkamat

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

BY THE NUMBERS

100	612	4,032	702	37.4%	99.9%
days of testing	production targets	reported findings	high or critical findings	of targets affected	of findings reachable

Abstract

Today, many teams write software faster than they can review it, which changes the types of security issues that reach production. Apex Black is an autonomous security system built for this challenge. It tests live web applications and APIs, develops attack hypotheses, and checks whether weaknesses can be exploited. This report covers its first quarter in use. Between 8 April and 17 July 2026, Apex Black produced 4,032 CVSS-scored findings across 612 production targets in software, financial technology, and enterprise organizations. Of these, 702 were high severity or above, and 124 were critical. At least one high or critical finding appeared on 37.4% of targets. Almost all findings were network-reachable (99.9%), and 93.4% did not need user interaction. The most common issues were exposed secrets and data, broken authentication, and broken access control. Fewer cases involved injection and remote code execution. Six redacted case studies show what these look like in practice, such as cross-account token theft, account takeover, and server-side template injection. These results show how often basic security boundaries are missed when software changes faster than teams can review 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 Black 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 Black 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 now release code much faster than before. While language models help speed things up, security issues go beyond who wrote the code. A feature might work as planned but still miss an authorization check, cross tenant boundaries, expose credentials, or accept input it should reject. These controls often rely on context outside the code or prompt that created the feature.

Reviewing this context takes time. Regular human reviews cannot catch every change on every live service, especially when many teams are working at once. Apex Black was created to provide continuous adversarial testing. Its autonomous agents map each target's attack surface, try to show real abuse, and send possible findings for further review. The system can test hundreds of targets at the same time.

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 Black system

Four parts of the workflow are important for understanding the results. Details about target selection, models, prompts, and adjudication are proprietary.

How the system works: Apex Black does not use signature scanning. Its agents map the exposed parts of a target, form hypotheses about how a target could be abused, and try to show real impact. A candidate only moves forward if the system can describe a clear abuse path and, when possible, test it.

How a candidate becomes a finding: The system starts with a broad search. Candidates go through an assessment stage, where they get a CVSS v3.1 score, and then a separate review stage. The 4,032 findings in this report met the internal threshold for reporting. Not all were confirmed by outside parties. Only the six cases in Section 7 include external review, and only when the vendor provided it.

How the system scales: Each target is tested in its own isolated environment, so many targets can be analyzed at the same time. This parallel approach made it possible to build such a large dataset in one quarter. The monthly totals in Section 4 show how much the system was used in each period and do not indicate a growth trend.

How severity is reported: Each finding has a machine-generated CVSS score. This report uses these bands: low (below 4.0), medium (4.0 to 6.9), high (7.0 to 8.9), and critical (9.0 and above). Using one scale keeps comparisons consistent. Section 10 discusses the limits of machine-assigned scores.

3. Study design and corpus

Scope. The dataset contains findings produced by Apex Black 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, Apex Black produced 4,032 CVSS-scored findings across 612 production targets. Of these, 702 (17.4%) were high severity or above (CVSS 7 and above), and 124 (3.1%) were critical (CVSS 9 and above). Unless stated otherwise, all 4,032 findings are included in the data that follows.

> **CORPUS INCLUSION NOTE**
>
> All findings reported in this paper are drawn from the system's internally adjudicated true-positive subset; false positives and unconfirmed candidates are excluded. This internal classification does not imply independent external confirmation.

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%), and only 1.3% were low severity. The high-impact group was smaller but still significant: 578 high and 124 critical findings. For teams deciding what to investigate, those 702 high or critical findings are important, even though medium-severity issues are most common.

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

The monthly numbers show how much Apex Black was used in each period. Since the run started on 8 April and ended on 17 July, April and July are partial months and not directly comparable to May and June. The number of findings varied across the quarter, with June lower than May. We include this breakdown for completeness, not to suggest a growth trend.

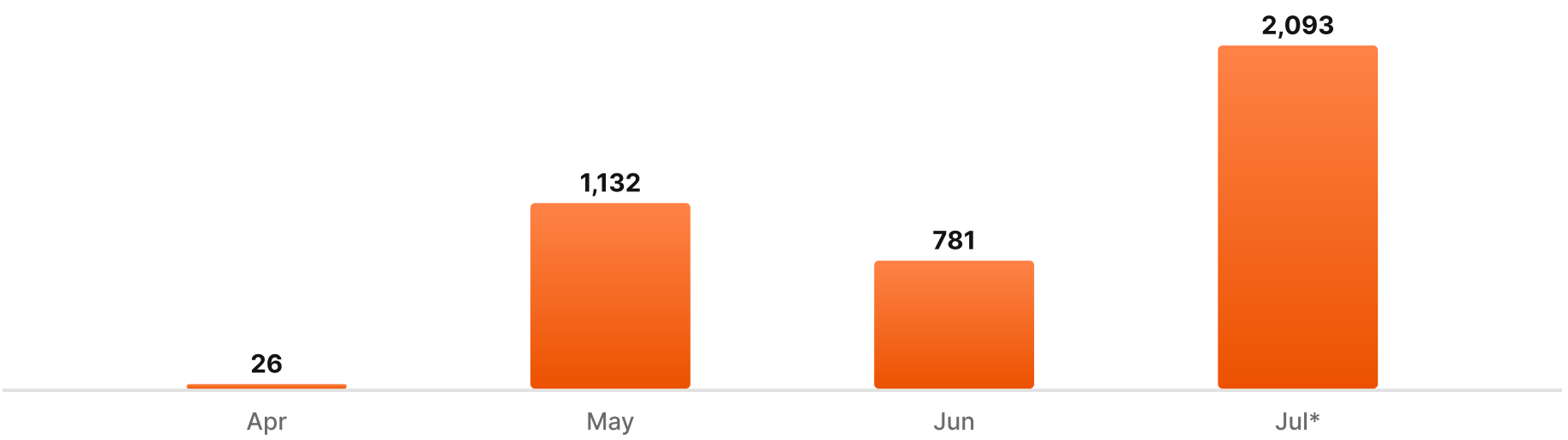


Figure 3. Findings by calendar month. April and July are partial months. Critical findings per month were 1, 46, 5, and 72 respectively. The breakdown reflects how much the system was run, not a growth trend.

5. Exploitability profile

CVSS severity measures both impact and exploitability. Three key metrics stand out: Attack Vector was Network for 99.9% of findings, User Interaction was None for 93.4%, and Privileges Required was None for 53.0%. This means remote and non-interactive weaknesses were common, and just over half did not need an authenticated user. These numbers are separate, so they do not show the exact combination for each finding.

93.4% needed no user interaction

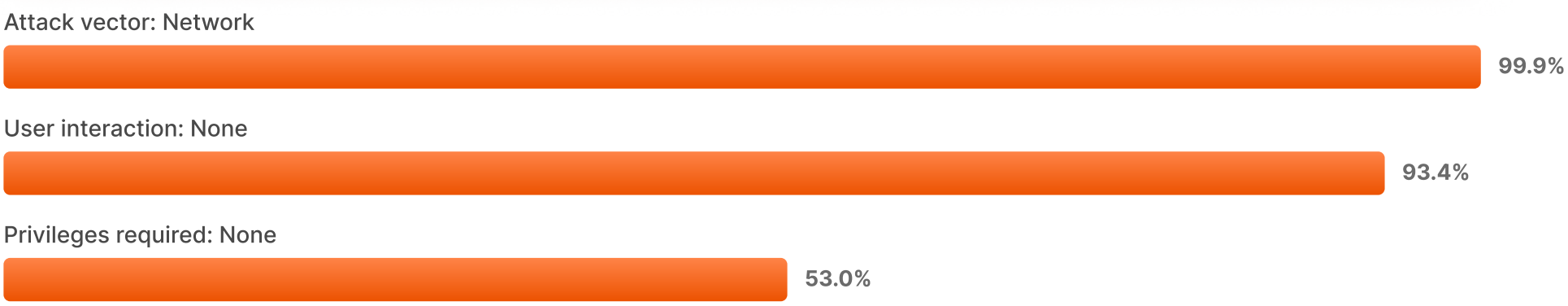


Figure 4. Exploitability profile over the 4,032 findings, each bar the share at the attacker-favorable extreme of one CVSS base metric.

This profile matches weaknesses at exposed boundaries, such as endpoints without authentication, resources returned without ownership checks, or secrets included in client code. Reachability metrics alone do not show root cause; the class distribution in Section 6 gives stronger evidence for these patterns.

6. A taxonomy of what breaks

Each finding is given one main class, so the counts do not overlap. About 18% of findings did not fit into a single class and are not shown in the chart.

The 3 largest categories primarily concern missing controls.

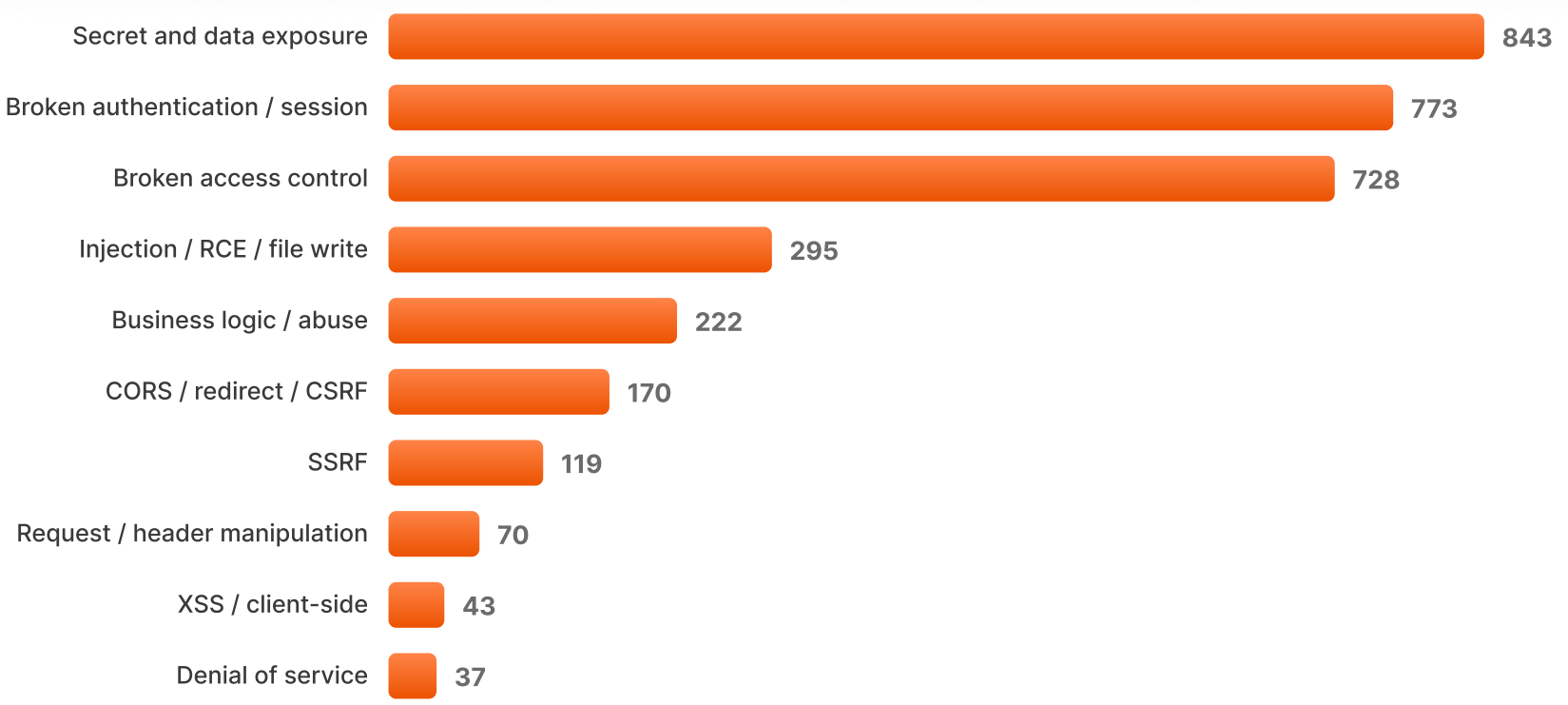


Figure 5. Vulnerability classes, one primary class per finding. About 18% did not map to a single class and are omitted. The three largest are all missing-control classes.

The three largest classes were secret and sensitive-data exposure (843 findings), broken or missing authentication (773), and broken access control (728). Together, they make up more than half of the classified findings. Each involves a security boundary that application code must keep intact, even as the code changes. Injection, remote code execution, and arbitrary file write made up 295 findings. This group was smaller but often had a bigger impact, since things like template engines or file handlers can turn untrusted input into server code. Section 7 gives examples. The other categories each made up less than 6% of the total findings.

7. Selected findings (redacted)

Aggregate numbers can make the real bugs seem abstract. The six cases below show what some high-impact findings looked like in practice. Each case gives only the target's sector and a paraphrased mechanism. The findings were reported to the affected organizations and, unless noted, reviewed or resolved by their security teams. This status applies only to these cases. Names, domains, endpoints, credentials, and exact steps are withheld.

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

A caching flaw served one user's live login token to the next person who opened the same link. No phishing, no interaction.

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

An authenticated API endpoint returned a live OAuth access token for the current user. A crafted suffix on the path parameter made the edge cache store that user-specific response under a cacheable key, so the next authenticated user who requested the same URL received the first user's token and could act as the victim against the identity API.

CVSS 7.4 (High). Disposition: resolved.

B. Victim-independent account takeover via HTTP parameter pollution

An attacker could take over any account by resetting its password, without the victim doing anything and without knowing a single secret.

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; another used them to decide 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, with no victim participation.

CVSS 9.1 (Critical).

C. Server-side template injection to remote code execution

A self-service report template became a way to run arbitrary commands on the vendor's own servers.

Enterprise reporting and 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 file reads and writes on the application server. An ordinary account was enough, with no stolen credential or victim interaction.

CVSS 9.9 (Critical).

D. Mass PII exposure via a leaked live API credential

A live payment-API key sitting in public documentation exposed hundreds of people's names, birth dates, and government IDs, and could move money.

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. It allowed paginated access to hundreds of payee records, including names, dates of birth, government identifiers, and addresses, and could mint session tokens for arbitrary users and initiate cross-program payment operations.

CVSS 9.3 (Critical). Disposition: resolved.

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

An anonymous file upload turned a routine document feature into full remote code execution. No account required.

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

One crafted chat message could run an attacker's code in every other user's session in the room.

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, so one message could run attacker code in each viewer's authenticated session and trigger account actions on their behalf.

CVSS 9.6 (Critical).

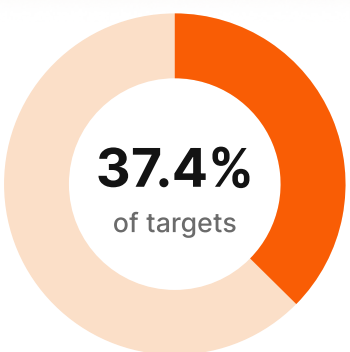
These cases cover different types of high-impact issues: two missing-authentication or control omissions (A and B), two code-execution paths (C and E), one secret exposure (D), and one client-side injection (F). All could be reached remotely, and none needed more than a regular account. They were chosen to show technical variety and impact, not to represent the whole dataset, where 81.3% of findings were medium severity.

8. Where risk concentrates

8.1 Concentration

Exposure was not the same across all targets. Of the 612 production targets, the median had 4 findings, while the most affected had 55. A few targets carried far more than most. More importantly, 37.4% of production targets had at least one high or critical (CVSS 7 and above) finding, so Apex Black reported at least one high or critical finding on more than a third of the applications it tested.

> **The most-affected single target carried 55 findings.**



4 median findings per production target

55 most-affected single target

702 high or critical findings in total

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 targets came from public and private bug-bounty programs and are mostly organizations that already invest in security review. Most are in financial technology and payments, developer and infrastructure tooling, enterprise identity and SaaS, and consumer application platforms. These were active production systems at organizations open to outside security research, not a random sample of all web applications, so the results describe this group and should not be read as a prevalence estimate for the internet overall.

8.3 What this says about AI-assisted development

The dataset does not track whether a person or a model wrote the vulnerable code. It cannot prove that AI-assisted development caused any specific finding, and it offers no pre-AI baseline. It does show a pattern that matters when teams use AI to speed up development:

1. **Composition.** The leading 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, and 53.0% needed no privileges (Section 5).
3. **Prevalence.** More than a third of production targets had at least one high or critical finding (Section 8).

These are common software-security issues, whether the code is written by hand or generated. AI tools can make them harder to manage because they let teams produce more code without adding time for threat modeling and review. A prompt to add an endpoint may not include the authorization rules, tenant checks, or secret-handling steps unless the team supplies that context and tests for it. The real risk is the gap between how fast code changes and how much review can keep up, no matter who wrote it.

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 needs to keep pace with delivery. Human judgment is still crucial, as shown in the vendor reviews behind the case studies. But regular, scheduled reviews cannot cover all changes when code is updated constantly across many services. Automated adversarial testing can help by expanding coverage and giving reviewers specific leads. Its value depends on careful filtering: a system that produces more candidates without filtering them just shifts the workload to the people who must investigate them.

Automation changes the balance for both defenders and attackers. Parallel agents can test more scenarios than a small team could check by hand, but attackers can use the same tools. Defenders benefit most when they run this testing on their own systems, connect it to fixing issues, and act before attackers find the same weaknesses.

What defenders can take from this

1. **Test the unhappy paths.** Most of these findings survive because functional tests authenticate as the intended user and send well-formed requests. Add negative, cross-tenant, and adversarial cases: requests from the wrong account, duplicated parameters, and inputs that should be rejected.

2. **Make review ongoing, not just periodic.** A scheduled review cadence cannot keep up with daily changes across many services. Review has to run alongside delivery, not after it.

3. **Connect testing to remediation.** Findings reduce risk only when they reach an owner with enough context to act. A pile of unadjudicated candidates just moves the bottleneck.

4. **Run it against yourself first.** The same automation is available to attackers; the advantage goes to whoever finds the path first.

10. Reading these numbers

There are several limits to how these numbers should be interpreted. First, bug-bounty targets are self-selected. They might have stronger security programs than average, but they could also have more complex attack surfaces, so this sample cannot estimate how common vulnerabilities are in all software. Second, severity, reachability, and class labels come from Apex Black's own process; this report does not estimate false positives, compare with other methods, or independently review every finding. Vendor review or resolution applies only to the six cases in Section 7, not to every label or CVSS score. Third, each finding is given one main class even if more could apply, and about 18% did not fit the shown categories. Finally, changes in monthly numbers reflect how much the system was used, not changes in the amount of vulnerable software, and both April and July are partial months.

11. Conclusion

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

These results show a real problem for teams using faster development tools: review capacity does not automatically keep up with more code. Continuous adversarial testing can help close this gap if it provides clear evidence, filters its results well, and stays connected to human review and remediation. That is what Apex Black is designed to do.