Eliminating Vulnerability Classes at Scale in the Post-Mythos Era

Jeevan Jutla
Gecko Security - August 2026

Executive Summary

> This brief is a written version of a talk given at Black Hat 2026, https://youtu.be/9rUiF4vyN3Q

Most of what a security team calls "work" is not solving new problems. It is solving the same problem over and over again, and the metrics almost everyone relies on are structurally blind to it.

Two of the most widely deployed open-source projects in the world disclosed 569 vulnerabilities between them. Fewer than 105 of those were actually distinct. We took every vulnerability n8n and GitLab had publicly disclosed, traced each one back to the commit that fixed it, and grouped findings that a single fix in one place would have prevented. Viewed that way, most of what both projects shipped as "new" turns out to be a repeat of something they had already patched elsewhere.

- **67% of n8n's disclosures and 58% of GitLab's were repeat issues.** The same underlying weakness, fixed again in a different file, kept showing up.
- **135 n8n advisories trace down to 32 root causes. 434 GitLab CVEs trace to 73.** What looked like an unmanageable backlog was really a short list of design decisions.
- The repeats alone **cost roughly $44k at n8n, and +$1M at GitLab** in bounties, plus 1,117 backports reapplying fixes that already existed.
- One **GitLab vulnerability class ran for 728 days**. 20 findings, 19 engineers, each independently writing the same one-line check, none knowing the last had already solved it.
- None of this shows up on a dashboard. MTTR and fix rate measure how fast findings close. A class fixed five times still scores as five clean fixes.

	n8n	GitLab
Total Vulnerabilities	135	434
Unique Vulnerabilities	32	73
Recurrence Rate (...explain)	67%	58%
$ Saved in avoiding recurrence payouts	~$44k	~$1.0m
Engineering time saved avoiding recurrence fixing	213 backports	904 backports

For most of the last decade, we measured the cost of all this in wasted engineering hours, and wasted hours are survivable. That is no longer the ceiling. Vulnerabilities are now exploited within hours of disclosure, and a model can reverse-engineer a patch into a

working exploit for the price of a coffee. Every patch you ship against a class you haven't actually closed is a map to the variants you left open.

The future of application security teams should stop counting how fast you close tickets and start counting how often you're closing the same thing twice. Recurrence rate tells you how much of your effort repeats a problem you've already solved. Vulnerability classes eliminated measures how often you've removed a vulnerability's root cause, not just the individual finding. When those two numbers become the target, the work changes shape: you stop chasing individual bugs and start removing whole categories of risk, fixing the design once, at a point every code path has to cross, and enforcing it so the class can't come back.

1. Why Now

For most of application security's history, the field rested on one assumption: finding a vulnerability was hard, and turning it into a working exploit was even harder. Risk was a function of how much time and effort an attacker was willing to spend, not of how many unresolved issues were sitting in your backlog. A backlog of mediums was a backlog of things that would never get weaponized or had minimal impact, so deferring them was rational.

That assumption has quietly stopped holding. In a UC Berkeley benchmark of nearly 900 real vulnerabilities, Claude Mythos Preview produced 157 working exploits, and GPT-5.5 produced 120, each one for a few dollars and a few minutes of compute. What used to require a strong security researcher is now something you can rent by the token. And the shift is already visible in breach data, not just benchmarks: for the first time in roughly two decades, stolen credentials are no longer the leading way attackers get in. Straightforward vulnerability exploitation has overtaken them, climbing from around 20 percent to 31 percent of confirmed breaches in a single year.

The Cloud Security Alliance CISO community, whose contributors include Jen Easterly, Bruce Schneier and Google's Heather Adkins, spelled out what this means for defenders, and the part that matters most here is uncomfortable. Every patch is now also an exploit blueprint. The same tools that find a bug can read your fix and recover the flaw it was hiding. So each time you patch one instance of a class, you're handing an attacker a labeled example of exactly the mistake to go looking for elsewhere in your code. Fixing a class one instance at a time no longer buys you the protection it used to.

2. Why Prioritization Stopped Working

No team has ever fixed everything, and that isn't a failure of any particular team; it's the scale of application security. So the industry built frameworks to safely decide what to ignore. CVSS, EPSS, and SSVC all do fundamentally the same job: they score a vulnerability on its own merits and tell you where it belongs in the queue. That worked when attackers were slow, because weaponizing a bug took months and almost nobody could do it, which made deferring the mediums a reasonable bet.

Microsoft SharePoint clearly illustrates what that bet costs today. In July 2025, two vulnerabilities were disclosed: an authentication bypass and an arbitrary file write. Scored individually, the bypass was "spoofing," and the file write required an authenticated session; one landed as medium severity, and the other as low; precisely the tiers a prioritized backlog is designed to defer. Chained together, they were a critical finding and gave an unauthenticated attacker remote code execution on the server, and that is exactly how they were used in the wild. Almost a year later, the same product saw fresh variants of the same bugs under active exploitation, reusing the same persistence technique.

Both halves of that story matter, because they fail in different places. The scoring failed because severity assessed one finding at a time cannot see a chain; a medium and a low finding that combine into a critical get filed for later. The remediation failed because each round of patches closed the specific bugs that were reported while leaving the design

decision that kept generating them untouched, which is why the variants came back a year later. When a model can find and chain vulnerabilities for a few dollars, the interesting question stops being *which handful of vulnerabilities we fix* and becomes *whether the class still exists once we've fixed the ones we can see*.

3. The Blind Spot in the Metrics

"Fix everything" sounds like a fantasy target, and it is a fantasy as long as the unit of work stays where it currently sits, at the individual ticket. It sits there because of how we measure the work.

Nearly every security team tracks mean time to remediate and fix rate. Both are a real improvement over simply counting open findings, but they share one blind spot: they measure activity, not whether risk is actually falling. And the failure mode is easy to miss precisely because it looks like success. You fix a vulnerability quickly this quarter, and your MTTR looks great. Months later, the same weakness resurfaces in another service, in a slightly different shape; your tooling files it as a brand-new finding with a fresh ticket and a fresh clock, and you fix it quickly again. The better your team gets at this rhythm, the healthier the dashboard looks, while the exposure underneath never moves. It’s Goodhart's law applied to security work: the moment a measure becomes the target, it stops measuring the thing you cared about.

4. Recurrence Rate

Recurrence rate answers the question the dashboard cannot: how much of what you’ve found this period,is just a variant of a class you've already fixed? A high rate means the team is spending most of its effort solving the same problems again.

The test for whether two findings belong to the same class is deliberately strict: one fix, in one place, would have prevented both. That is a much sharper boundary than grouping by a shared CWE label. Two findings both tagged "XSS" can have entirely different root causes and belong to different classes; an XSS and a path-traversal bug can belong to the *same* class if both trace back to the same missing control. The strictness is the point. Anyone can eyeball two findings and argue the grouping either way, and it's the single-fix test that turns this from an opinion into a measurement.

A second number follows naturally from the first. Classes eliminated counts the root causes you've closed for good, and unlike most security metrics, that number only moves downward and stays there. None of this replaces MTTR; the two sit side-by-side and ask different questions. MTTR asks how fast you closed the ticket. Recurrence rate asks whether closing it actually mattered. Fast MTTR alongside high recurrence rate is not a healthy program; it's a team closing tickets briskly while the underlying risk remains.

5. The Evidence and the Cost

We tested the idea against two projects with long, fully public security histories. From April 2025 and July 2026, n8n disclosed 135 GitHub security advisories; from January 2024 and July 2026, GitLab disclosed 434 CVEs. Of those 67%of n8n's disclosures and 58% of GitLab’s were variants of root causes exposed by earlier findings. Every finding was traced to a public advisory and commit and verified against the repository. If the finding couldn't be assigned to a class with confidence, we excluded it. These figures are therefore lower bounds, not estimates.

Grouped by root cause, the duplication is impossible to miss. n8n's 135 advisories reduce to 32 root causes; GitLab's 434 CVEs reduce to 73. That collapse is also what makes "fix everything" tractable. The raw counts describe the scale of the problem; the root-cause counts describe the problems that actually remain. No team can sustainably keep fixing 135 findings forever, but it can genuinely work through 32 design decisions.

The repeats carry a bill. n8n has paid out roughly $66,000 in bounties and GitLab around $1.7 million; at each project's own rates, variants of previously exposed classes account for about $44,000 and $1 million of that, respectively. On top sits the engineering cost: 213 backports at n8n and 904 at GitLab, each one requiring an engineer to reapply a fix for a problem the project already understood.

At n8n, a single prototype-pollution class produced nine findings in about four months, roughly one every fortnight. A safe function that operated correctly already existed in the codebase; none of the vulnerable paths called it. The fix was never a mystery. Nothing made it mandatory, and that is how recurrence survives even when the right control is sitting right there.

When GitLab returns a list, it checks whether the user can access that list, but not whether you're allowed to see each object inside it. The fix is a single line. You confirm the current user can read the object before returning it. But that line had no shared home, so every engineer writing a new endpoint had to remember it from scratch. Its absence produced 20 findings over 728 days, fixed across 77 commits by nineteen different engineers, each writing the same check in isolation, none aware the last one had already solved it. Most were rated medium severity, the tier a prioritized backlog is built to defer, and the tier SharePoint just showed chaining into a critical.

Underneath both failures is the same mechanic. In each project, roughly half of all fixes were applied where the reported vulnerability appeared rather than at a shared point every code path has to cross, which leaves sibling vulnerabilities open by design. That gap is exactly what recurrence rate makes visible, and it's why the classes kept coming back. GitLab runs the more mature of the two programs, and the data reflects it: a lower recurrence rate, classes that resurface more slowly. But the two year class above emerged out of that mature program. Maturity slows the treadmill. It doesn't get you off it.

6. What Changes

Getting off the treadmill comes down to four shifts, none of which require a new budget line.

1. The unit of work moves from the ticket to the class. You're no longer working a bug; you're working the family it came from, and the job isn't done when the reported instance closes.

2. The fix moves from the reported line to the design decision. Instead of patching wherever the researcher happened to look, you trace the group back to the choice that allowed all of it and fix it once, at a point every code path must cross. In this study, about half of all fixes landed on the reported line instead, which is precisely why the siblings stayed open.

3. Grouping moves from the CWE label to the root cause. Labels describe the symptom. The only test that matters is whether one fix, in one place, would have prevented both findings.

4. The fix moves from a one-time change to a rule enforced on every commit. This shift does the actual elimination, because the next engineer who reintroduces the class gets stopped at the door instead of getting caught by a researcher eighteen months later. Across the two projects, that meant 23 rules for n8n and 65 for GitLab. This is what "classes eliminated" looks like as a number rather than a slogan.

The best teams already do a version of this without naming it: pick a class, rally the engineers behind it, put a ninety-day clock on it, burn it down, and add a check so it cannot return. The method isn't the missing piece. What's missing is a measurement that tells you whether it worked.

7. About Gecko

The hard part was never knowing the fix. It was doing this at the scale a real codebase demands, tracing a group of findings to one design decision, finding every variant, and writing the rule that holds.

Gecko was built to group findings by root cause rather than by label, trace each group to the decision underneath, hunt the variants still open across the codebase, and write the check that stops the class returning on the next commit. None of it depends on the tool. The method works with a spreadsheet and a disciplined team, and the data here is public and reproducible. What changed is the economics. When finding and weaponising a bug costs a few dollars and a few minutes, fixing each instance by hand is no longer affordable, and closing the class is the only thing that scales.

See which classes are still open in your code. https://gecko.security/

Appendix A. The Recurrence Data

For each disclosure we pinned the fix commit and verified it against the repository. We diffed the code before and after the patch, then traced the path from the attacker controlled input to the dangerous operation. A finding joined a class only if one fix, at a single shared point in that path, would have prevented every member. Uncertain findings were dropped rather than counted, so every figure is a lower bound. The source data is public GitHub Security Advisories for n8n and public CVEs with their fix commits for GitLab.

Metric	n8n	GitLab
Corpus	135 GitHub advisories	434 CVEs
Window analysed (fix release date)	Apr 2025 to Jul 2026 (about 15 months)	Jan 2024 to Jul 2026 (about 29 months)
Recurrence rate	67% (90 of 135)	58% (251 of 434)
Root cause classes	32	73
Median recurrence half life*	49 days	105 days
Bounties paid, all findings**	~$66,000	~$1.74M
Recurrence tax (variants only)**	~$44,000	~$1.0M
Fix commits (total rework)	318	1,229
Backports (shipping the same fix again)	213	904

** The median across classes of the mean gap between consecutive findings in a class, that is, how often a class produces its next variant.*
*** Modelled at each project's own published bounty rates (n8n up to $1,000 per bug; GitLab up to $35,000 per critical), applied to the disclosed severity mix. The recurrence tax is the share of that total attributable to variants of classes already fixed.*

Appendix B. The Complete Dataset

All 569 disclosures, each assigned to a single root cause class. n8n: 135 findings across 32 classes; GitLab: 434 across 73. Classes run largest first, singletons last. Each carries two fields, Root cause (the design decision that produced the family) and Enforce (the commit-time rule that would prevent recurrence). Every ID and fix commit resolves to its public source.

(link to where it appears on website)