REVERSINGLABS

The Patch Is the Attack Payload

Decades of patch management got us here: Our diffs are being read by AI — and our patches have become the adversary's answer key, writes Mario Vuksan, CEO and co-founder of ReversingLabs.

Mario Vuksan
CEO AND CO-FOUNDER, REVERSINGLABS

Foreword

Twenty-three years ago, we knew who the enemy was in a way that felt almost comforting: a lone actor with a virus or worm, betting that most of the internet hadn't gotten around to installing a patch that already existed. In 2003, the SQL Slammer worm proved that bet was correct, spreading globally in a matter of minutes and causing massive disruptions by exploiting a patch that had been disclosed and patched months earlier.

Since then, much of our enterprise security architecture has been built to correct for failures like that. Vulnerability management — the scanners, bulletins, the Patch Tuesdays and compliance regimes — treated deployment as the hard problem, and for a long time, we were right.

I've spent my career focused on the other side of that patching problem: not "did you install it," but "what does this patch actually do?" ReversingLabs was founded, in part, to address the fact that software provenance and behavior aren't perfectly aligned. A signed binary tells you who vouched for it. A manifest or software bill of materials (SBOM) tells you what should be inside it. But neither tells you what is actually inside that software — or what it will do when it is deployed.

That gap, between what software claims and what it is, has shaped nearly everything we've built here at ReversingLabs, advocated for, and worried about publicly and privately.

This piece started because I spent the summer of 2026 watching my own industry do something remarkable, and I think under-examined. In seven weeks, we stood up four vulnerability clearinghouses — two commercial, one non-profit and one federal — that all point at the same target: flaws in the world's open-source software. The clearinghouses seek to address those flaws at a scale and speed nobody has attempted before. It is genuinely good work and should continue.

But a question I couldn't stop asking myself was this: "What does a patch look like from the other side? Not to the defender receiving it, but to the adversary reading it?"

When we start publishing those fixes at industrial scale, in public, on a corpus anyone can read, aren't we also telling the world exactly where the holes in open source are, in language a machine can act on in minutes, for about a dollar? And that steady stream of intelligence is certain to drive down the mean time to exploit (MTTP) for flaws in open source.

That will highlight an uncomfortable fact, that our current information security industry is built around a metric that is fast losing its relevance: how quickly organizations identify and patch their flaws. To be clear, this isn't an argument against patching. It is an argument that MTTP stopped being a useful control variable somewhere around the point at which the number of things to patch became unbounded, even as function-level reachability analysis found that previous vuln scanning tooling was flagging flaws that were over 97% unreachable by malicious actors — and therefore not a threat.

And then there's the elephant lurking at the network edge: the software binaries powering appliances that are the starting point of a disproportionate share of real intrusions, but get almost no attention. The firewalls, VPN gateways, and industrial controllers organizations rely on run closed-source, proprietary software that is vulnerable — but also unknowable to security teams.

This piece explains how the whole, complex picture of software supply chain risks is bigger than finding flaws in open source and locking down software provenance. The key is being able to attest to the actual behavior of the software we produce, consume and deploy. I've believed that for a long time. Events of this past summer just made it harder to ignore.

Introduction

Three hundred and seventy-six bytes. That was the whole worm. SQL Slammer had no payload, no persistence, no exfiltration, no ambition beyond arithmetic: generate a random IP, send yourself there, repeat. In 10 minutes it reached roughly 90% of every vulnerable host on the internet. Routers folded. ATMs went dark. Emergency call centers in Washington state started taking notes on paper. [1]

The focus of the attack wasn't unknown. Microsoft shipped the patch in July 2002. MS02-039. Six months earlier. The vulnerability had been found by David Litchfield, who demonstrated proof-of-concept code at Black Hat, dutifully reported it, and watched it get fixed.[2]

And then Slammer went through Microsoft's own network, because Microsoft had not finished installing Microsoft's patch.[3]

In the early 2000s, the cybercrime world was in its infancy. There was no ransomware installed, nor data stolen by Slammer. Like many attacks at the time (The Anna Kournikova virus), SQL Slammer was designed purely to make noise and get attention - and that's it. Still, the trauma of SQL Slammer's disruptions was real and the lesson seemed obvious: We knew the bug existed, had a fix available for it, but hardly anybody applied it. The consequences of that failure were easy to see.

It was one of the foundational traumas that shaped the information security industry. And the industry (and market) responded. Everything we built for the next 23 years — the entire multibillion dollar apparatus of vulnerability management — was a response to threats like SQL Slammer. The core problem, we concluded, was patch deployment.

We were right — for a while. But we also created a trap that our industry is walking into right now with tremendous enthusiasm and excellent intentions: using frontier AI models to try to slam shut the patching window that has hung open for decades: finding, fixing, forking, and publishing patches for the world's open-source software at industrial scale. But we have not stopped to ask what a patch looks like from the perspective of the bad guys.

It looks like an answer key.

How We Learned to Love the Patch

Here is a detail worth sitting with: The CVE program was not born out of source code review. It was born out of asset scanning.

In January 1999, at the 2nd Workshop on Research with Security Vulnerability Databases at Purdue University, two MITRE engineers, David E. Mann and Steven M. Christey, presented a paper called "Towards a Common Enumeration of Vulnerabilities."[4] The motivation section is almost startlingly unglamorous. MITRE was running a pile of network assessment tools and intrusion-detection systems from a wide variety of vendors on its own infrastructure, each with its own implicit vulnerability database, and nobody could tell whether Vendor A's finding and Vendor B's finding were the same bug.[5]

A working group formed. It became the 19-member CVE Editorial Board. The original list had 321 entries and went public in September 1999. Eight years later, the asset half of the same idea got its own standard in CPE, so that it could name the software it found with the same rigor that the CVE brought to naming the flaw. Enumerate the IT estate, enumerate the defects, join the two tables, assign an owner. That is vulnerability management. It is a beautiful, boring, structural idea. And it worked.

27 Years of Defensive Advances, and Their Offensive Inputs

Year	Defensive Advance	The Offensive Input It Produced
1999	CVE launches with 321 entries	A common noun for a bug, born out of asset scanning, not code review
2002	Trustworthy Computing memo	Code Red and Nimda were the input to the Trustworthy Computing memo.
2003	Slammer, followed by the first Patch Tuesday	A six-month-old patch; the calendar becomes infrastructure; Exploit Wednesdays, inevitably
2004	Structural comparison of executable objects	BinDiff — changing code is cheap; detecting the change is expensive
2008	Automatic patch-based exploit generation is possible	Working exploits from Windows Update patches; fastest in under 30 seconds
2010	npm ships	Ownership is solved by abolishing it; the patch queue becomes unbounded
2024	Mean time to exploit crosses zero	-1 day; exploitation now routinely precedes the patch
2026	Four clearinghouses in seven weeks, then BOD 26-04.	Frontier models patch the commons at machine speed, in public

Figure 1: Each defensive advance produced a durable offensive input. The 2026 entries are the first where the trade-off is visible in advance.

The Zine Era, and Why Nobody Bothered Diffing Anything

The primary literature of this period was not academic. It was Phrack and Bugtraq and 2600. In 1996, Phrack Issue No. 49[6] carried Aleph One's "Smashing the Stack for Fun and Profit," which taught an entire generation how a stack overflow becomes a shell and which remains the most consequential piece of security pedagogy ever published outside a university.

Larry Cashdollar tells a story about this era that captures it perfectly. He was poking at a program called midikeys and inadvertently changed the root password on an SGI Onyx graphics system — at the exact moment the engineers were demonstrating that Onyx to a Navy admiral. Somebody had to walk into the demo room and hand the engineers the new password. Cashdollar's career survived. The bug became his first CVE.[7]

And the CVE process, as he describes it, was gloriously informal: "You didn't file for a CVE; you published it on a security mailing list," and someone at MITRE who thought your finding was worthy would assign an identifier. People on the list would test your work, argue with you, verify it. Peer review by mailing list.

Notice what nobody was doing in this period: reversing patches.

Why would you? The bugs were lying in the street. Unauthenticated RPC overflows, format strings, off-by-ones in daemons running as root, CGI scripts that would cat any file you named. If you wanted a vulnerability, you did not need to reverse a vendor's hotfix to find one; you needed an afternoon and a copy of the source tarball, or not even that. The binary was never the bottleneck. Deployment was the bottleneck.

So the industry optimized for the bottleneck, and the bottleneck was people not installing things.

January 2002: The Memo

Code Red and Nimda in 2001 did what no security team's slide deck could. In January 2002, Bill Gates sent the Trustworthy Computing memo to every Microsoft employee, calling for products "as available, reliable and secure as standard services such as electricity" — and the four pillars that followed reorganized how the largest software company on earth thought about shipping code.[8]

Then, 12 months later, Slammer happened anyway, on a six-month-old patch. Which brings us to October 14, 2003, and the first official Patch Tuesday: seven bulletins, five of them rated critical. Christopher Budd, who helped launch Patch Tuesday, explained the choice of Tuesdays with an honesty you rarely get from a vendor — it was "the earliest day in the week that we felt we could sustainably offer these."

And so the calendar became infrastructure. Adobe eventually aligned to the same second Tuesday of each month. Enterprises built change windows around it. Mean time to patch became a board metric, a compliance artifact, a line item in a hundred thousand contracts, and eventually an entire industry.

There is a footnote to Slammer that matters enormously for the rest of this paper. Tom Gallagher, who now runs the Microsoft Security Response Center, has said that one of the things that came out of Slammer was "rethinking about publishing exploit code." The lesson the vendor took from a worm built on public proof-of-concept code: Publish less.

Hold on to that. It comes back.

The Scorecard: 23 Years of Mean Time to Patch (MTTP)

We should be honest about how the strategy performed, although the answer is uncomfortable and the industry mostly declines to state it.

27 Years of Defensive Advances, and Their Offensive Inputs

Case	Detail	Gap
SQL Slammer	MS02-039, July 2002 to January 25, 2003	6 months
WannaCry	March 2017 to May 2017, 150+ countries	2 months
2018–19	Attacker mean time to exploit	63 days
2023	Outlier-adjusted	5 days
2024	Exploitation precedes disclosure	−1 day
2025 (est.)	Attacker mean time to exploit	−7 days

Figure 2: The gap collapses across 23 years and then inverts.
Horizontal scale is compressed; the last two rows sit to the left of the patch, before it exists.

Read the bottom of that table again. Mandiant measured mean time to exploit at 63 days across 2018–19, 32 days across 2021–22, and five days in 2023.[9] In M-Trends 2026, the figure crosses zero: -1 day for vulnerabilities disclosed in 2024, with an estimated -7 days for 2025.[10] Exploitation now routinely happens before the patch is public.

Attacker Mean Time to Exploit vs. Defender Mean Time to Remediate

Period	Series	Days
2018–19	Attacker time to exploit, after disclosure	63 days
2021–22	Attacker time to exploit, after disclosure	32 days
2023	Attacker time to exploit, after disclosure	5 days
2024	Attacker time to exploit, before disclosure	−1 day
2025 (est.)	Attacker time to exploit, before disclosure	−7 days
Flat for years	Defender time to remediate a critical vulnerability	55 days

Figure 3: The attacker series crosses zero in 2024. The defender series has been roughly flat at 55 days for years.

Meanwhile, the defender's side of the ledger has barely moved. Independent measurement puts mean time to remediate a critical vulnerability at roughly 55 days, and it has been roughly flat for years.[11] WannaCry ran on a two-month-old patch in 2017; Slammer ran on a six-month-old patch in 2003. Fourteen years of the discipline bought us a four-month improvement in the worst case.

And the installed base is worse than the calendar suggests. Black Duck's 2026 audit of 947 commercial codebases found that only 7% of open-source components were running the latest version, that 92% of codebases contained components four or more years out of date, and that 93% of codebases contained components with no development activity for at least two years.[12]

The security industry is built around a metric that measures how fast we start, on a population where most of the work never finishes.

None of which is an argument against patching. It is an argument that MTTP stopped being a useful control variable somewhere around the point where the number of things to patch became unbounded. Which happened for a specific reason.

How We Acquired 100,000 Owners — and None at the Same Time

The vulnerability management model assumed that software has a vendor, that the vendor has a support contract, that the contract has a patching obligation, and that your organization has a person whose job it is to apply it. CVE plus CPE plus an owner. Clean.

Then we invented package managers and all that stopped being true.

When the Registries Arrived

Year	Registry	Origin/Builder
1995	CPAN	Perl community; the proof that a language-level registry could work at all
Early 2000s	Maven	As an Apache Turbine sub-project/Jason van Zyl, later founder and CTO of Sonatype
2003	PyPI	Originally a pure index without hosting/Python community, informally the Cheese Shop
2004	RubyGems 0.2.0	March 14, released on Pi Day/Chad Fowler, Jim Weirich, David Alan Black, Paul Brannan, Richard Kilmer
2008	pip	October 15/Ian Bicking, as an alternative to easy_install
2010	npm	January 12/Isaac Z. Schlueter

Figure 4: 15 years of language-level registries, and the people who built them. None of them was sold as a security architecture.

The npm origin story is the one to dwell on, because it tells you exactly what these systems were optimized for. The first registry was a JSON file inside a Git repository — since lost to time. That was quickly replaced with a CouchDB application in which anyone could publish without authentication and overwrite anyone else's packages.[13] npm was Schlueter's side project for about four years, running on donated infrastructure, before the scale of it forced him to start a company.[14]

Nobody sold these things as a security architecture. Nobody had to. The pitch was pure developer velocity: Do not write it — declare it. And it worked so well that it ate the world.

Open source is now effectively universal. Black Duck found the code in 98% of the 947 commercial codebases it audited, with roughly 70% of all scanned code being of open-source origin.[15] More than 90% of the Fortune 500 depends on it.[16]

Which means we solved the ownership problem by abolishing ownership. There is no vendor to call about a transitive dependency four levels down that a solo maintainer stopped touching in 2019. There is no contract. There is no patching mandate. There is, frequently, no maintainer.

The Bill Arrives: Transitive Math and Unreachable Bloat

In 2019, Zimmermann, Staicu, Tenny, and Pradel published, at USENIX Security, the measurement that should have changed the conversation: Installing an average npm package implies trust in roughly 79 other packages and 39 maintainers.[17]

What One Dependency Actually Costs

Measure	Value
Packages implicitly trusted by the one package you chose	79
Maintainers implicitly trusted by the one package you chose	39
Direct dependencies per package, 2011 to 2018	1.3 to 2.8
Transitive dependencies over the same period	~80
Packages reached by each of the top five	134,774–166,086

Figure 5: A small linear increase in what developers choose produces a superlinear increase in what they inherit.

The mechanism is worth understanding because it is not linear. Direct dependencies per average package crept from 1.3 in 2011 to 2.8 in 2018 — a small, sane, defensible increase. Over the same period transitive dependencies reached about 80. A small linear increase in what developers choose produces a superlinear increase in what they inherit. And it runs in both directions: An average package reached about 230 others by 2018, while the top five packages each reached between 134,774 and 166,086.[18]

Up to 40% of all npm packages depended on code with at least one publicly known vulnerability. And the paper notes something crueler still: For more than half of npm packages, no patch was available at all.[19]

Then the AI Coding Assistants Arrived

Black Duck's 2026 OSSRA report is the clearest picture we have of what happened next, and it reads like a stress test result:[20]

- Mean open-source vulnerabilities per codebase: 581 — up 107% in a single year, the first doubling in the report's history
- Open-source components per codebase: up 30% year over year
- Files per codebase: up 74% year over year, to over 84,000 — quadrupled in five years
- Codebases containing at least one vulnerability: 87% (78% contained high-risk vulnerabilities; 44% contained critical ones)
- Surveyed organizations that experienced a software supply chain attack in the preceding year: 65%

So the patch queue grew by 107% in 12 months. Now here is the part that makes the whole edifice absurd.

Most of It Doesn't Matter, and Finding Out Which Part Does Matter Is the Actual Work

Roughly 95% of vulnerabilities live in transitive dependencies — code the developer never chose.[21] And when you analyze whether the vulnerable function is callable from the application, most findings evaporate. Function-level reachability analysis routinely cuts findings by around 92%, and vendors report customers whose previous tooling was flagging vulnerabilities that were over 97% unreachable. The academic literature[21] on dependency bloat and debloating — DepClean, JShrink, and the Maven-ecosystem studies — documents the same phenomenon from the other end: A great deal of what we ship is never executed.[22]

The Patch Queue, Filtered

Filter Step	Share
Mean open-source vulnerabilities per codebase	581
In transitive dependencies that the developer never chose	~95%
Not callable from the application	~92%
The reachable, deployed remainder	What is left

Figure 6: MTTP does not distinguish between the item that matters and the 92% that do not.

Put those two findings side by side and the operational picture is grim in a very particular way. We have built a workload where:

- The median item has no security value, because the code is unreachable;
- The upgrade carries real stability risk, because it is a version bump in a transitive dependency you did not choose and cannot test in isolation;
- The volume is unbounded and growing at over 100% a year; and
- The metric we are graded on — MTTP — does not distinguish between the item that matters and the 92% that do not.

This is the adverse effect the industry has been slow to say out loud: open-source bloat converted vulnerability management into a denial-of-service attack against engineering: Time that could go to new code, to architecture, and to reachable exposure goes instead to servicing a queue generated by a dependency graph nobody designed.

The Federal Government Just Conceded the Point

On June 10, 2026, the U.S. Cybersecurity and Infrastructure Security Agency (CISA) issued Binding Operational Directive 26-04, "Prioritizing Security Updates Based on Risk."[23] It supersedes and revokes both BOD 19-02 and BOD 22-01 — the entire preceding federal patching regime — and replaces calendar-and-CVSS deadlines with a four-variable risk matrix:[7]

Is the asset publicly exposed?

Is the flaw in the Known Exploited Vulnerabilities catalog?

Can exploitation be automated?

What is the post-exploitation technical impact?

Meet all four and you have three calendar days plus mandatory forensic triage. Meet some and you have 14 or 60. Meet none and the vulnerability is deferred to the next scheduled system upgrade.[24]

CISA states the rationale plainly: Attackers' growing use of AI is collapsing the interval between a patch shipping and a working exploit existing. Agencies were given 60 days to update their processes and until December 7, 2026, to meet the full timelines, with the Federal Risk and Authorization Management Program (FedRAMP) aligning its own rules to the same date.[25]

This is the U.S. government formally abandoning "patch everything" and adopting reachability-style prioritization by directive. Exposure and automatability are reachability questions wearing a policy hat. The argument this paper makes about bloat is no longer contrarian; it is binding policy for federal agencies.

DORA's measurements land in the same place from the delivery side. In 2024, each 25% increase in AI adoption was associated with a 1.5% decrease in delivery throughput and a 7.2% decrease in delivery stability. In 2025, throughput turned positive as teams adapted — and the stability finding held.[26] More change, more breakage. DORA's own framing is that AI amplifies whatever practice you already had.

4 Clearinghouses in Just 7 weeks

Now we have the frontier models, and they are extremely good at finding vulnerabilities.

Anthropic's Frontier Red Team reported finding and validating more than 500 high-severity vulnerabilities in open-source software.[27] The team was explicit about why it started there: Open source runs everywhere, and much of it is maintained by small teams or volunteers with no dedicated security resources. IBM cited a figure of nearly 3,900 high- or critical-severity open-source vulnerabilities identified by Anthropic's Mythos Preview model. DARPA's AI Cyber Challenge, run from 2023 to 2025, demonstrated the thing that actually matters: seven finalist cyber-reasoning systems that confirm a vulnerability with a proof of vulnerability and then synthesize a patch validated against that proof.[28]

Immediately, and predictably, the maintainers drowned. The curl project shut down its bug bounty after AI-written submissions overwhelmed reviewers with unconfirmed findings.[29] FFmpeg maintainers criticized Google for reporting valid bugs without supplying patches, calling the reports "CVE slop." When AI automation stops at discovery, the burden lands on volunteers.

The answer, arriving in a remarkable seven-week burst and culminating in a federal program, was the clearinghouse.

Seven Weeks in the Summer of 2026

Program	Design and reported scale
Project Lightwell IBM + Red Hat May 28, 2026	Trusted enterprise clearinghouse; commercial subscriptions. Lightwell Network generally available; Clearinghouse Premier in limited availability for financial services, with a controlled channel for patch embargoes. Scale reported: $5B commitment, 20,000+ engineers; 6,500+ remediated, signed, and certified Java and Python dependencies at launch.
Athena Chainguard-led coalition June 15–16, 2026	Clearinghouse pooling and correlating member findings; stacks independent protection layers where a clean patch does not yet exist. Scale reported: 20,000 findings and 2,000+ patches across 500 projects at launch; 40,000+ vulnerabilities within three weeks, 42% critical or high, 86% network reachable.
Akrites Linux Foundation June 25, 2026	Shared Security Incident Response Team plus a single standardized coordinated disclosure process; confidentiality-first; maintainer of last resort for critical unmaintained packages. Scale reported: Founding commitments from AWS, Anthropic, Chainguard, Cisco, Citi, Endor Labs, Ericsson, Google, IBM, JPMorganChase, Microsoft and GitHub, NVIDIA, OpenAI, RapidFort, Red Hat, Rust Foundation, Sonatype, Vodafone, and Zscaler.
Gold Eagle The White House July 14, 2026	Federal AI cybersecurity clearinghouse under Executive Order 14409. Housed in Treasury with DHS through CISA, the Department of War through the NSA, and the National Cyber Director. Built on VINCE, operated with Carnegie Mellon University's Software Engineering Institute. Voluntary for industry. Scale reported: Directed on a 30-day clock and launched on day 42; reported to be already ingesting and prioritizing vulnerability data across sectors, coordinating scanning verification, and pushing remediation. Focused heavily on open source.

Figure 7: Four clearinghouses — three commercial, one federal — were announced in under seven weeks, every one of them pointed at open source.

Gold Eagle is the one that changes the character of the exercise, because it is the government joining in. Executive Order 14409, signed June 2, 2026, and titled "Promoting Advanced Artificial Intelligence Innovation and Security," directed the Treasury Department — in consultation with the National Cyber Director, the Secretary of War through the NSA, and DHS through CISA — in voluntary collaboration with the AI industry and operators of critical infrastructure.[30]

It launched on July 14. National Cyber Director Sean Cairncross described the design as enabling "vulnerability and patching coordination at a speed and scale never seen before."[31] The intake mechanism is VINCE — the Vulnerability Information and Coordination Environment — run with Carnegie Mellon's Software Engineering Institute, which means anyone can report into it. The executive order specifically names rural hospitals, community banks, and local utilities as the operators who should benefit.[32]

And Gold Eagle, too, focuses heavily on open source. Four clearinghouses, three commercial and one federal, and every one of them pointed at the same corpus.

Four Clearinghouses, One Corpus, and the Loop That Closes Behind Them

Stage	What it holds
The four clearinghouses	Project Lightwell (IBM + Red Hat), Athena (Chainguard coalition) and Akrites (Linux Foundation), all voluntarily feeding Gold Eagle (the White House)
The open-source commons	Find, fix, fork, and publish, at machine speed; readable by everyone, equally
Validated, localized, security-relevant diffs	Source patches, forks, patched historic versions, and regression tests demonstrating the triggering input
The adversary, reading the answer key	Pre-localized search space; a working exploit in 10 to 15 minutes at roughly $1 per attempt
Out of frame: third-party binaries, containers, appliances, firmware	No repository to canvass; 44% of zero-days in 2024 targeted enterprise technologies, 20 of those 33 in security and networking products, and exploitation has been the leading initial infection vector for six consecutive years

Figure 8: The industry's most powerful analytical capability is concentrated on the one corpus where the adversary can reproduce our results exactly.

A dated policy cliff is approaching fast that deserves more attention than it is getting. The protections against information-sharing liability under the Cybersecurity Information Sharing Act of 2015 were authorized only through September 30, 2026, and were extended to December 11, 2026 by the Continuing Appropriations and Extensions Act, 2027, signed on September 2, 2026.[33] Every organization submitting proprietary vulnerability data into a federal or industry clearinghouse right now is doing so under protections that expire on December 11. That is not a reason to stay out. It is a reason to read the data-sharing agreement before signing it.

The Cloud Security Alliance's research note on Gold Eagle reaches a conclusion worth putting in front of anyone building on it: Operators should treat clearinghouse-sourced remediation guidance as an input to existing change-management and patch-validation processes rather than as a substitute, given the absence of published criteria for safe patch deployment in operational environments — and they should build the capacity to reconcile multiple, potentially conflicting prioritization signals rather than assuming that one authoritative source will emerge.[34]

That is, nearly word for word, recommendation No. 2 of this paper, arrived at independently.

Chainguard's Dan Lorenc deserves credit for voicing the least promotional sentence uttered by any vendor all summer: "Fragmentation is worse, standing still isn't survivable."[35] He is right. Fragmentation would be worse. Akrites requiring that reporters submit a patch after validating a finding is genuinely good design. Lightwell's upstream-always posture, submitting fixes back to originating communities, is the correct instinct. Palo Alto Networks' bolting virtual patching onto Lightwell is a sensible acknowledgment that some exposures cannot be fixed in the time available.

This is a good development. It is also, structurally, four clearinghouses — three commercial and one federal — built on a public commons. And the reason that every one of them chose that commons is worth stating precisely.

Why Open Source and Not the Applications That Get Breached?

There are five reasons, and only three of them get mentioned in press releases.

- **Access:** You can read it. A frontier model canvassing open source needs no contract, no NDA, no customer. First-party code belongs to someone else. Third-party code arrives as a binary.
- **Leverage:** One fix in a widely used library protects thousands of downstream consumers — the best security return per dollar in the industry, measured that way.
- **Benchmarkability:** Repositories are a substrate you can score. AIxCC ran on open source because it had to. Careers, funding, and research agendas follow whatever can be measured, and repositories can be measured.
- **The ownership vacuum:** Nobody objects. There is no vendor to negotiate with, no EULA to breach, no license term forbidding analysis, no relationship to damage. You can fork a library. You cannot fork an F5 appliance.
- **And the uncomfortable one:** Open source is the only corpus on which a frontier lab can demonstrate large-scale security capability in public. That is not a criticism of anyone's motives. It is an observation about incentives, and incentives determine where the effort goes.

Which produces the defining asymmetry of this era, and it runs the wrong way.

Open source is equally available to defenders and to attackers. First-party and third-party code is not. We have concentrated the industry's most powerful analytical capability on the one corpus where the adversary can reproduce our results exactly.

The Two-Tier Disclosure Problem

Now the harder question about clearinghouses, and it is a question rather than an accusation.

A clearinghouse that gives subscribers early access to fixes is, by construction, a two-tier disclosure regime. Lightwell Clearinghouse Premier is described as a controlled channel for patch embargoes and threat coordination, initially for financial services providers.[36] Athena stacks protections around members while a durable upstream fix is pending. Both approaches are defensible. Both are also, from the outside, indistinguishable from having a period in which a fix exists, is deployed to paying customers, and is not yet public.

Two problems follow.

Two Tiers, One Exposure Window

Party	What Happens Between Day 0 and Day 40
Subscriber	Receives the fix on day 0, then validates, integrates, regression-tests, and rolls out. Deployed on day 40, exposed for 40 days.
Everyone else	Has no advisory, no CVE, no severity score, and therefore no ticket.
Adversary	Reads the fix. Obtains it from a fork, a container image or a build artifact, in 15 minutes at a cost of roughly $1.

Figure 9: Silent patching does not buy defenders time. It transfers time from the defender to the attacker.

The first problem is that near-zero-time deployment is not achievable, even for subscribers. The fix has to be validated, integrated, regression-tested, rebuilt, and rolled out across an estate. Black Duck's own data says only 7% of components are on their latest version. DORA says pushing more change faster costs stability. A subscriber who receives a patch on day 0 and deploys it on day 40 is exposed for 40 days — during which the patch exists in a fork, in a container image, in a build artifact — somewhere an adversary can obtain it.

The second problem is everyone else. Nonsubscribers face exploits derived from fixes for which there is no advisory, no CVE, no severity score, and therefore no ticket. They are not merely behind. They are blind, and the thing blinding them is a defensive measure.

This is not hypothetical, and it is not new. It is the oldest trick in the book, and it has a literature.

Binary Diffing: The Oldest Trick, Newly Subsidized

In 2003, a German mathematician who signed his work Halvar Flake stood up at Black Hat Europe and gave a talk called "Graph-Based Binary Analysis." The following year, at DIMVA in Dortmund, Germany, Thomas Dullien published "Structural Comparison of Executable Objects" — his first academic paper, and the origin of BinDiff.[37]

The insight was that you should not compare bytes. Bytes move when the compiler feels like it. You should compare structure: Build the control flow graph of both binaries, fingerprint the functions, match them across versions, and the functions that fail to match are the ones whose logic changed. In 2005, with Rolf Rolles, Dullien published the improved version at SSTIC.[38]

Buried in the 2004 paper is the sentence that this entire white paper exists to restate. Dullien observed that changing source code and recompiling takes relatively little work, while "the analysis of the object code will have to be completely redone."

So the asymmetry was named, in print, 22 years ago, by the man who built the tool. Cheap to change. Expensive to detect. We have spent two decades making the cheap part cheaper.

The tooling arrived immediately behind the theory. SABRE Security shipped BinDiff commercially in 2004. In 2006, eEye built its own Binary Diffing Suite, used internally to analyze Microsoft's Patch Tuesday releases. Why? Jeongwook "Matt" Oh put it plainly in his Black Hat USA 2009 slides: Patch analysis was "the only way to obtain some secret information they don't release."[39] The following year, he presented "ExploitSpotting: Locating Vulnerabilities Out of Vendor Patches Automatically."[40]

Read that 2009 slide again. In 2006, the security industry was already using patch diffing as an intelligence source specifically to recover information that vendors declined to publish. Silent patching was identified as an exploitable asymmetry 20 years ago, by defenders, and laid out in public at Black Hat in 2009.

Then in 2008, David Brumley, Pongsin Poosankam, Dawn Song, and Jiang Zheng presented "Automatic Patch-Based Exploit Generation Is Possible" at IEEE Security and Privacy.[41] Given a program and its patched version, generate an exploit for the vulnerability the patch fixes — automatically. Using patches from Windows Update, they produced working exploits in many cases within minutes; their fastest end-to-end time to a verifiable exploit was under 30 seconds. Their framing of the result is the most quotable line in the field: "A fundamental tenet of security is to conservatively estimate the capabilities of attackers."

The community even named the phenomenon. The day after Patch Tuesday has been called Exploit Wednesday for roughly two decades. We have had a folk term for this problem since before the iPhone came out.

Is Diffing Cheaper Than Mining? Wrong Question — but the Answer Matters

The claim to test is whether binary diffing is cheaper than canvassing open source yourself with open-weight models. Stated that way it is not quite the right comparison, and the precise version is more damning.

Mining a Corpus Yourself vs. Diffing a Released Patch

	Mining a Corpus Yourself	Diffing a Released Patch
Inputs required	Corpus selection, buildable environments, fuzzing or analysis harnesses, compute budget	Two artifacts and a graph matcher
Search space	The whole program, unlabeled	Pre-reduced to the functions the vendor changed
Triage burden	Large false-positive population; you must prove reachability and exploitability yourself	The vendor already localized, triaged, and validated; shipping the change is the vendor asserting it is security-relevant
What you learn	A candidate bug	A confirmed bug, its location, and its guard condition
Marginal cost per additional target	High; each program is a new search	Near zero; each new release is another labeled example

Figure 10: Diffing is not cheaper in some absolute sense. It is pre-localized — a categorically different economic activity.

And we now have a price. The CSA, analyzing what it calls the collapsing exploit window, found that AI systems can generate working exploits for disclosed CVEs in roughly 10 to 15 minutes at a cost of about $1 per attempt.[42] In the same analysis, 32.1% of newly tracked exploits in 2025 appeared on or before the public disclosure date of the CVE, and up 8.5 percentage points from 2024.

One dollar. Fifteen minutes.

Against a patch that a defender spent weeks validating, integrating, regression-testing, and rolling out.

Diffing is not cheaper in some absolute sense. It is pre-localized. You are not searching; you are reading an answer key that someone else compiled, validated, and published. That is a categorically different economic activity, and its cost scales with the number of patches in the world rather than with the difficulty of the underlying bugs.

And there is a strictly cheaper variant. Diffing a source patch requires no control flow graph recovery, no symbol reconstruction, no filtering of compiler noise. It names the function and states the condition, and it frequently ships with a regression test demonstrating the triggering input. Everything that made Dullien's 2004 problem hard is simply absent.

The clearinghouse era publishes source patches and patched historic versions of millions of components at machine speed. That is the cheapest input to exploit generation that has ever existed, produced at industrial volume, by defenders, on purpose.

Think Like the Adversary: Four Gifts We Are Wrapping for Them

Everything above is context. Here's the argument.

Agentic patching is a good development and should continue. The following four things are what a competent adversary would want us to keep doing while we do it.

	Practice	Why It Helps the Adversary
01	We publish the answer key, at scale, continuously.	Every patch is a specification of the bug it fixes. Every fork is a labeled example.
02	We are codifying silent patching.	It is a fix with no advisory, generates no ticket, no urgency, and no budget, and it sits in a public commit log.
03	We do not analyze binaries.	SolarWinds, XZ Utils, and 3CX were all invisible to source review.
04	We are not looking at third-party appliances.	That is where intrusions start and where no clearinghouse can canvass.

Figure 11: Four practices a competent adversary would want the industry to keep.

Gift 1: We Publish the Answer Key, at Scale, Continuously

Analyzing, patching, and forking open source with frontier models produces diffable release pairs at machine speed. Every patch is a specification of the bug it fixes. Every fork is a labeled example. Every patched historic version widens the set of components for which a functional, validated, security-relevant diff is publicly obtainable.

Brumley's numbers were from 2008, from binaries, without machine learning. The 2026 version of that pipeline consumes source diffs and has a frontier model attached.

Gift 2: We Are Codifying Silent Patching

Clearinghouses produce secure components faster than the disclosure machinery can describe them. Premier tiers with embargo channels formalize the interval. And the practice was already endemic before AI touched it: A research literature exists purely to detect silent vulnerability fixes, and an analysis of more than 600 open-source projects found over 100 security fixes announced in release notes with no associated CVE.[43]

The asymmetry is total. Attackers mine commits and diff releases, which the literature shows is tractable and automatable. Downstream defenders wait for advisories and prioritize by severity score. A fix with no advisory generates no ticket, no urgency, and no budget — while sitting in a public commit log.

Silent patching does not buy defenders time. It transfers time from the defender to the attacker. Matt Oh said so, with slides, in 2009.

Gift 3: We Do Not Analyze Binaries

The three compromises that most shaped supply chain policy were all invisible to source review.

- **SolarWinds:** The company's own investigation states that the threat actor did not modify the source code repository and that the malicious activity occurred within the automated build environment for the Orion Platform.[44] CrowdStrike's analysis of the SUNSPOT injector describes a loop that watched for MSBuild processes and substituted source files before the compiler read them, with safeguards to keep the injected lines out of build logs.[45] Roughly 18,000 organizations installed a validly signed result.
- **XZ Utils (CVE-2024-3094):** The macro that triggered the backdoor build existed in the release tarballs and not in the Git repository. The second-stage artifacts sat in Git as inert test files.[46] Red Hat's advisory notes that the injection was only included in full in the download package.[47] Repository code review would not have caught it.
- **3CX:** A signed, trojanized desktop application distributed through the vendor's own update channel.[48] Again: valid signature, clean-looking source, compromised artifact.

Reproducible builds are genuine progress here — Debian made them a hard requirement for the Debian 14 cycle in May 2026, with 98.29% of architecture-independent packages reproducing.[49] But reproducibility proves that a binary matches its source. It does not prove that either is benign. It is a tampering control, not a behavioral one. And it took a volunteer project roughly a decade to require reproducible builds in a far more homogeneous environment than any enterprise pipeline.

Gift 4: We Are Not Looking at Third-Party Appliances

Ask an incident responder where intrusions actually start and you will hear about edge devices. F5. Fortinet. Ivanti. The security and networking products that Typhoon-class actors have made their home.

Where the Analysis Is Pointed, and Where the Artifact Ships

Stage and Analysis	What It Covers
Code — SAST	Source defect analysis
Code — SCA	Component license and vulnerabilities
Code — FUZZ	Unknown vulnerability discovery
Build — the binary that ships	Complete component and binary analysis
Run / QA — DAST	External runtime scanning
Run / QA — IAST	Internal runtime monitoring
Where clearinghouse and agentic-patching effort goes	The code stage
The artifact both sides can see	The binary that ships
Where exploitation is observed	The binary that ships

Figure 12: The mandates are converging on the binary from one side while the analytical investment goes to the repository from the other.

Google's Threat Intelligence Group found that 44% of zero days exploited in 2024 targeted enterprise specific technologies, with a pronounced focus on security and networking products such as VPNs and firewalls.[50] Mandiant has reported exploitation as the leading initial infection vector for six consecutive years, at 32% of investigated intrusions.[51]

The sharpest illustration is BOD 26-04's own first real-world application. The day after the directive took effect, CISA added CVE-2026-10520 to the KEV catalog with a three-day deadline: a maximum severity unauthenticated OS command injection in Ivanti Sentry, allowing remote code execution as root on publicly exposed instances. It met all four risk criteria.[52] The Shadowserver Foundation discovered 19 vulnerable instances in its own scans, at least two of them already backdoored, and warned that its visibility was limited by blocked scans and that any unpatched instance should be presumed compromised.[53]

The first vulnerability that the most aggressive federal patch mandate in history was written to catch was in a third-party security appliance. Not a library. Not a package. An appliance, with no repository for any clearinghouse to canvass, and one the scanning community judged was probably compromised anywhere it had not been patched.

These appliances are largely built on open source. They ship as opaque binaries and firmware images. There is no repository to canvass, so the clearinghouses cannot canvass them, so the industry's most powerful new capability is pointed almost entirely away from where the exploitation is occurring.

We are canvassing the corpus we can read, at enormous expense, while the adversary works the corpus nobody is reading.

Why Source Code? Because It's Easy, Legal, and Legible

The honest answer to "Why is everyone focused on source code analysis?" has three parts, and none of them is "Because that is where the risk is."

Tractability
Source is text. Text in, text out. Git is ground truth, tests are an oracle, and a language model was trained on it. Binary analysis requires unpacking nested and arbitrary container formats, resolving statically linked and vendored code, recovering structure from stripped executables, separating compiler-introduced variance from deliberate modification, and doing all of it inside a CI budget. It is a different engineering discipline, and it does not fall out of giving an agent repository access.

Legality and Liability
You may fork and patch an open-source library without asking anyone. Analyzing a commercial appliance's firmware runs into license terms, anti-circumvention questions, vendor relationships, and the awkward fact that the vendor — not you — carries the contractual patching obligation. The path of least legal resistance leads directly to the commons. The EU Cyber Resilience Act (CRA), in force from December 2024, puts obligations on manufacturers and therefore points at the shipped artifact rather than the repository.

And the mismatch now runs in two directions at once. The CRA points at the artifact. BOD 26-04 points at prioritization. Gold Eagle points at coordination. All three are correct, and not one of them requires anybody to examine the artifact that actually gets deployed. The mandates are converging on the binary from one side while the analytical investment goes to the repository from the other, and nothing in between verifies that the fix a clearinghouse validated is present in the build a defender shipped.

Legibility
Repositories are benchmarkable. Benchmarks attract research, research attracts funding, funding attracts engineers. There is no SWE-bench for firmware.

Add these together and the industry's revealed preference is clear: We chose the corpus with the most transparency and the least accountability. Which is, inevitably, the corpus where our defensive work is most perfectly reproducible by the adversary.

One more honest note about capability. A survey of frontier AI's impact on the security lifecycle assessed attack detection as demonstrated in the real world and triage as demonstrated in research — while remediation development and deployment showed no demonstrated effect at scale.[54] The industry has automated finding vulnerabilities. It is automating fixing. It has not automated deploying, which was the original problem from January 2003.

What to Prepare For

Reprioritize around what the adversary actually does, not what is convenient to measure.

	Move	What It Means
1	Focus on the entry point.	Analyze what actually shipped and what actually runs at your edge.
2	Address the velocity coming out of the clearinghouses.	Patched components are unverified artifacts, not trusted fixes.
3	Use binary differential analysis to find the silent patches.	What actually changed, and is the promised fix present in the artifact you deployed?
4	Maximize patching, minimize waste.	Integrate detection with reachability signal across source and binary workflows.
5	Measure the estate, not the calendar.	What fraction of deployed artifacts contain the fix, verified at the artifact level?

Figure 13: Five moves, in order of how much exposure each removes.

1. Focus on the Entry Point

Third-party and open-source applications as delivered — binaries, containers, appliances, firmware, installers — are where intrusions begin, and they constitute the one artifact class nobody in the current architecture examines. If you analyze one thing, analyze what actually shipped and what actually runs at your edge.

2. Address the Velocity Coming out of the Clearinghouses

Patched components arriving from Lightwell, Athena, Akrites, Gold Eagle, a vendor fork, or your own agentic pipeline are unverified artifacts, not trusted fixes. Treat them accordingly. Verify what you are handed, before you deploy it, at the artifact level. A signed patch from a reputable clearinghouse is a provenance claim; it is not a behavioral guarantee.

3. Use Binary Differential Analysis to Find the Silent Patches

If the industry is going to patch faster than it discloses — and it is — then defenders need the telescope the attackers have had since 2004. Differential analysis of successive builds answers two questions the release notes will not: “What actually changed?” and “Is the fix you were promised actually present in the artifact you deployed?” Run it on inbound third-party components, on your own build outputs, and on the appliances at your perimeter.

4. Maximize Patching, Minimize Waste

BOD 26-04 has already made this federal policy for civilian agencies, with exposure and automatability as two of its four variables; everyone else should follow the same logic. Integrate vulnerability detection with reachability signal across both agentic source workflows and binary workflows. Roughly 92% of findings are not callable. Every hour spent on those is an hour stolen from the reachable, deployed exposure that will actually be used against you. This is not an argument for patching less. It is an argument for spending the patching budget on the subset that exists in the attacker’s ranked list rather than in yours.

5. Measure the Estate, Not the Calendar

MTTP measures how fast you start. The adversary attacks what you never finished. Replace or supplement it with unpatched-estate metrics: The fraction of deployed artifacts that contain the fix, verified at the artifact level, not at the ticket level. Only 7% of components are on their latest version. That number, not your MTTP, is what the adversary is working from.

Coda

Every defensive advance in this field has produced an offensive input. We standardized vulnerability names and gave attackers a shopping catalog. We regularized patch releases and gave them Exploit Wednesday. We published proof-of-concept code and taught Slammer's author how to write it.

None of those were mistakes. The alternative — secrecy, incoherence, no common noun for a bug — was worse. But we have generally noticed the trade-off somewhere around the second or third worm.

This time, the trade-off is visible in advance. We are about to produce validated, localized, security-relevant source diffs for millions of components, continuously, at machine speed, on the one corpus our adversary can read exactly as well as we can. Halvar Flake handed out the tool at Black Hat Europe in 2003 and told us in print in 2004 that changing code is cheap and detecting the change is expensive.

We have spent 22 years making that first half cheaper and almost nothing on the second half. The going rate for the result is now about $1.

Fix the components. Absolutely fix the components. But look at the binary that comes out the other end, because that is the only artifact both sides can see — and right now only one side is looking.

Provenance attests to origin. Only analysis attests to behavior.

References

[1] https://www.wired.com/2003/07/slammer/

[2] SQL Slammer technical accounts: patch MS02-039 (CVE-2002-0649) released July 2002; worm observed January 25, 2003; 376 bytes; approximately 90% of vulnerable servers reached within 10 minutes. Proof-of-concept origin attributed to David Litchfield, Black Hat Briefings.

[3] The Register, "MS Struggles to Contain Worm," January 28, 2003, reproducing internal Microsoft release-management memos during the Slammer outbreak. Microsoft, "Statement on the Slammer Worm Attack," January 25, 2003.

[4] MITRE / CVE Program, "CVE — History." Concept presented January 21–22, 1999 at the 2nd Workshop on Research with Security Vulnerability Databases, Purdue University; 321 original entries; public launch September 1999.

[5] Mann, D.E., and Christey, S.M., "Towards a Common Enumeration of Vulnerabilities," MITRE, 1999. See the motivation section on integrating findings from multiple network assessment tools and intrusion detection systems.

[6] https://phrack.org/issues/49/smashing-the-stack-for-fun-and-profit_md#article

[7] Dark Reading, "An Uncommon 20 Years of Commonly Enumerating Vulns," 2019, including Larry Cashdollar's account of the midikeys/SGI Onyx incident and of early CVE assignment practice via Bugtraq.

[8] Gates, W., Trustworthy Computing memo, January 2002. See also Microsoft Security Blog, "Celebrating 20 Years of Trustworthy Computing," 2022.

[9] Mandiant/Google Cloud, time-to-exploit analysis of 138 vulnerabilities exploited in 2023: mean TTE 63 days (2018– 19), 32 days (2021–22), five days (2023, outlier-adjusted; 47 days including outliers). October 2024.

[10] Mandiant, M-Trends 2026: mean TTE reported as -1 day for vulnerabilities disclosed in 2024, with an estimated -7 days for 2025.

[11] Edgescan vulnerability statistics, as compiled in ProjectDiscovery, "The Vulnerability Curve Bent with the AI Curve," 2026: defender time to remediate a critical vulnerability approximately 55 days and roughly flat.

[12] Black Duck, 2026 Open Source Security and Risk Analysis (OSSRA) report, February 2026. Based on 947 commercial codebases across 17 industries: 98% contain open source; mean 581 vulnerabilities per codebase, up 107%; components up 30% year over year; files up 74% year over year to more than 84,000; 87% contain at least one vulnerability, 78% high-risk, 44% critical; only 7% of components on latest version; 92% of codebases contain components four or more years out of date; 68% of components more than two years old; 65% of surveyed organizations experienced a supply chain attack.

[13] Roleke, K., "npm: How Did We Get Here?" on the original npm registry as a JSON file in a Git repository, its CouchDB successor, and unauthenticated publishing.

[14] Open Source Underdogs, episode 38, interview with Isaac Z. Schlueter on npm's origins as a four-year side project on donated infrastructure.

[15] Nesbitt, A., "Package Manager Timeline," 2025: npm first release January 12, 2010 (Isaac Z. Schlueter); RubyGems 0.2.0 released March 14, 2004 (Chad Fowler, Jim Weirich, David Alan Black, Paul Brannan, Richard Kilmer); pip introduced October 15, 2008 (Ian Bicking); Maven, created by Jason van Zyl as an Apache Turbine sub-project; PyPI launched 2003 as a pure index.

[16] IBM Newsroom, "IBM and Red Hat Commit $5 Billion to Redefine the Future of Open Source in the AI Era," May 28, 2026, including the >90% Fortune 500 open-source dependency figure and the cited Anthropic Mythos Preview figure of nearly 3,900 high- or critical-severity open-source vulnerabilities. IBM also used data from worldmetrics.org, a statistics aggregator.

[17] Zimmermann, M. et al., "Small World with High Risks: A Study of Security Threats in the npm Ecosystem," USENIX Security 2019. Average package implies trust in ~79 packages and 39 maintainers; direct dependencies 1.3 (2011) to 2.8 (2018) against ~80 transitive; average package reach ~230, top five 134,774–166,086; up to 40% of packages depend on code with a known vulnerability; more than half of npm packages have no available patch.

[18] Zimmermann, M. et al., "Small World with High Risks."

[19] Zimmermann, M. et al., "Small World with High Risks."

[20] Endor Labs, State of Dependency Management and product documentation: approximately 95% of vulnerabilities exist in transitive dependencies.

[21] Endor Labs, function-level reachability analysis: approximately 92% reduction in SCA findings, with reported customer cases above 97% of flagged vulnerabilities not reachable. Vendor-reported figures from customer data rather than independent benchmark.

[22] Soto-Valero, C. et al., studies of bloated dependencies in the Maven ecosystem and the DepClean tool; Bruce et al., JShrink. See also "The Used, the Bloated, and the Vulnerable: Reducing the Attack Surface of an Industrial Application," arXiv:2108.05115.

[23] CISA, Binding Operational Directive 26-04, "Prioritizing Security Updates Based on Risk," issued June 10, 2026. Supersedes and revokes BOD 19-02 (2019) and BOD 22-01 (2021). Four variables: asset exposure, KEV status, exploit automation, post-exploitation technical impact.

[24] CISA, BOD 26-04 Implementation Guidance, and Cloud Security Alliance research note on BOD 26-04: tiered remediation timelines of three, 14, or 60 days, with mandatory forensic triage in the three-day tier and deferral to the next scheduled upgrade where exposure criteria are not met.

[25] Tenable, "What Is CISA BOD 26-04," June 2026, on the 60-day process update and 180-day (December 7, 2026) full-compliance milestones; FedRAMP Public Notice 0014, June 2026, aligning FedRAMP Vulnerability Detection and Response rules to the same date.

[26] Tenab DORA, Accelerate State of DevOps Report 2024: each 25% increase in AI adoption associated with a 1.5% decrease in software delivery throughput and a 7.2% decrease in delivery stability.

[27] Anthropic Frontier Red Team, "Zero-days" research post: more than 500 validated high-severity vulnerabilities found in open- source software, with stated rationale for beginning with open source.

[28] DARPA AI Cyber Challenge (AIxCC), 2023–2025. Seven finalist cyber-reasoning systems, each confirming vulnerabilities with a proof of vulnerability and synthesizing a patch validated against it. See also "OSS-CRS: Liberating AIxCC Cyber Reasoning Systems for Real-World Open-Source Security," arXiv:2603.08566.

[29] Reported in OSS-CRS: The curl project's bug bounty shutdown following AI-written submissions, and FFmpeg maintainers' criticism of unpatched AI-generated reports.

[30] Executive Order 14409, "Promoting Advanced Artificial Intelligence Innovation and Security," signed June 2, 2026. Section 2(d) directed Treasury, in consultation with the National Cyber Director, the Secretary of War through NSA, and DHS through CISA, to establish a voluntary AI cybersecurity clearinghouse within 30 days.

[31] Cybersecurity Dive, "US Launches Vulnerability Clearinghouse Amid AI-Fueled Surge in Flaws," July 15, 2026, and The Record, "Trump Administration Unveils AI-Supported Clearinghouse for Cyber Vulnerabilities," July 15, 2026, quoting National Cyber Director Sean Cairncross. Gold Eagle announced on July 14, 2026.

[32] Cybersecurity Dive on VINCE — the Vulnerability Information and Coordination Environment — operated in partnership with Carnegie Mellon University's Software Engineering Institute, and on Gold Eagle's stated focus on open-source software.

[33] K&L Gates law firm via National Law Review, and Cloud Security Alliance research note on Gold Eagle: Information-sharing protections under the Cybersecurity Information Sharing Act of 2015 authorized only through September 30, 2026 absent reauthorization.

[34] Cloud Security Alliance, research note on the Gold Eagle AI vulnerability clearinghouse, July 2026: Recommendations to treat clearinghouse-sourced remediation guidance as an input to existing OT (operational technology) and ICS (industrial control system) change-management and patch-validation processes rather than a substitute, and to build capacity to reconcile multiple potentially conflicting prioritization signals.

[35] Infosecurity Magazine, "Chainguard, BNY Team Up to Secure Open Source from AI Threats," June 16, 2026, quoting Dan Lorenc. See also Chainguard, "Athena" and "Expanding Athena," and PR Newswire, July 7, 2026: More than 40,000 vulnerabilities processed within three weeks, 42% critical or high, 86% network-reachable.

[36] IBM, Lightwell product pages, and the Silicon Review, July 9, 2026: Lightwell Network generally available with over 6,500 remediated, signed, and certified Java and Python dependencies; Lightwell Clearinghouse Premier in limited availability for financial services as a controlled channel for patch embargoes and threat coordination. Linux Foundation, "Linux Foundation and Industry Leaders Launch Akrites," June 25, 2026.

[37] Flake, H. (Dullien, T.), "Structural Comparison of Executable Objects," DIMVA 2004, pp. 161–173. See also "Graph-Based Binary Analysis," Black Hat Europe 2003, and "More Fun with Graphs," Black Hat Federal 2003.

[38] Dullien, T., and Rolles, R., "Graph-Based Comparison of Executable Objects," SSTIC 2005. Author's own account at thomasdullien.github.io.

[39] Oh, J. ("Matt"), "Fight Against 1-Day Exploits: Diffing Binaries vs Anti-Diffing Binaries," Black Hat USA 2009, including the eEye Binary Diffing Suite (2006) used internally for Patch Tuesday analysis.

[40] Oh, J., "ExploitSpotting: Locating Vulnerabilities Out of Vendor Patches Automatically," Black Hat USA 2010.

[41] Brumley, D., Poosankam, P., Song, D., and Zheng, J., "Automatic Patch-Based Exploit Generation Is Possible: Techniques and Implications," 2008 IEEE Symposium on Security and Privacy, pp. 143–157. Fastest reported end-to-end generation of a verifiable exploit: under 30 seconds. See also Zhou, J. et al., "Finding a Needle in a Haystack: Automated Mining of Silent Vulnerability Fixes," ASE 2021, pp. 705–716; Zhou, J. et al., "CoLeFunDa: Explainable Silent Vulnerability Fix Identification," ICSE 2023, pp. 2565–2577; and VFDelta, arXiv:2409.16606.

[42] Cloud Security Alliance, "The Collapsing Exploit Window": AI systems generating working exploits for disclosed CVEs in approximately 10 to 15 minutes at roughly $1 per attempt; 32.1% of newly tracked exploits appearing on or before the CVE's public disclosure date.

[43] Chainguard, "Vulnerability Fixes in Plain Sight: How Your Scanners Are Missing Hundreds of Vulnerabilities," June 2024: analysis of more than 600 Wolfi projects identifying over 100 security fixes with no associated CVE.

[44] SolarWinds, "An Investigative Update of the Cyberattack" and Security Advisory FAQ: The threat actor did not modify the source-code repository; malicious activity occurred within the automated build environment for the Orion Platform.

[45] CrowdStrike, "SUNSPOT Malware: A Technical Analysis," January 2021.

[46] thesamesam et al., "xz-utils Backdoor Situation (CVE-202 4-3094)": Published release tarballs did not contain the same code as the Git repository; the build-to-host.m4 macro in the tarballs differed from upstream.

[47] Red Hat and NSFOCUS advisories on CVE-2024-3094: The malicious injection was obfuscated and only included in full in the download package.

[48] FortiGuard Labs, "3CX Desktop App Compromised (CVE-2023-29059)," 2023.

[49] Debian Release Team, debian-devel-announce, May 2026: Reproducible builds made a hard requirement for the Debian 14 "Forky" cycle; 98.29% of architecture-independent packages reproducing (23,731 passing, 414 flagged) at enforcement.

[50] Google Threat Intelligence Group, zero-day analysis for 2024: 75 zero-days exploited in the wild, 44% targeting enterprise-specific technologies with emphasis on security and networking products.

[51] Mandiant, M-Trends 2025: exploitation the leading initial infection vector for the fifth consecutive year, accounting for 33% of investigated intrusions.

[52] CVE-2026-10520, unauthenticated OS command injection in Ivanti Sentry, added to the CISA KEV catalog with a three-day remediation deadline days after BOD 26-04 took effect.

[53] Shadowserver Foundation scanning identified 19 exposed instances, two already backdoored.

[54] "Frontier AI's Impact on the Cybersecurity Landscape," arXiv:2504.05408: Lifecycle assessment placing attack detection as demonstrated in the real world, triage as demonstrated in research, and remediation development and deployment as showing no demonstrated effect.

About RL

ReversingLabs is the trusted name in file and software security. We provide the modern cybersecurity platform to verify and deliver safe binaries. Trusted by the Fortune 500 and leading cybersecurity vendors, RL Spectra Core powers the software supply chain and file security insights, tracking over 422 billion searchable files daily with the ability to deconstruct full software binaries in seconds to minutes. Only ReversingLabs provides that final exam to determine whether a single file or full software binary presents a risk to your organization and your customers.

REVERSINGLABS

Worldwide Sales +1.617.250.7518
sales@reversinglabs.com