Hypothesis, Not Syntax: The Human-AI Boundary in Linux Kernel Vulnerability Discovery

Ștefan-Daniel Wagner, Eduard-Cristian Popovici, Laurențiu Boicescu and Traian-Cristian Ureche

***Abstract*—The Linux kernel assigns a Common Vulnerabilities and Exposures (CVE) identifier to nearly every bugfix, making its rising CVE count a reporting-policy artifact, not a capability signal. Underneath that noise sit two primary-verified kernel bug families. A single 2023 commit introduced two epoll race conditions: an AI system found the first, its fix silently removed the signal that might have flagged the second, and the second, a 6-instruction race reachable from a browser sandbox, waited for a human. The other, a deterministic page-cache flaw that an AI-scaled search found in about an hour once a human framed the hypothesis, resurfaced in a second subsystem three weeks later through a manual audit. We verify both families against primary sources and, classifying an independently supplied list of 86 CVE identifiers, build a four-row evidence table and a funnel of what we set aside. In both families the boundary falls at hypothesis novelty, not syntactic complexity or a race, exposing a dual-use asymmetry: the capability that closes out a known pattern is not confined to defenders. We argue for two investments, not one, plus two narrower practices: re-auditing a fix's neighborhood and funding CVE triage.**

***Index Terms*—vulnerability discovery, large language models, kernel security, AI-assisted auditing, dual-use risk, defensive security**

I. Introduction

The Linux kernel assigned more than 4,300 CVE identifiers in 2024 alone [1]. Read naively, that number looks like a security crisis, or, if AI-assisted auditing gets the credit, like proof that AI has transformed vulnerability discovery. Neither reading survives contact with how the number is produced: the kernel's own CVE team assigns an identifier to almost every bugfix, regardless of whether the bug was ever exploitable [2]. Section II sets out exactly how large that reporting-policy effect is. Underneath the noise, however, 2026 also produced a small number of individually verified, root-capable kernel bugs whose discovery stories are genuinely informative, and they do not tell a simple story.

The clearest natural experiment sits in the Linux kernel's epoll code. A single 2023 commit introduced two latent race conditions. Anthropic's Mythos model found the first, in the course of a defensive AI-assisted review program. The second, a 6-instruction race nicknamed "Bad Epoll" that roots Linux and Android and is reachable from inside Chrome's renderer sandbox, went unfound until a human researcher working through Google's kernelCTF program reported it, after the first bug's own fix had quietly removed the dynamic-analysis signal that might have flagged it. A second natural experiment followed a researcher's account of finding a deterministic, non-race logic flaw nicknamed "Copy Fail," by forming a hypothesis himself and then using an AI tool to search for it at scale: within three weeks the same invariant was independently rediscovered in a different kernel subsystem, by a team that used no automated tool at all.

Together, these two families let us ask a sharper question than "can AI find vulnerabilities": where, specifically, does AI assistance change who finds a bug, and where does it not. Our answer is that the boundary tracks whether a searchable hypothesis about the bug already exists, not whether the bug is a race condition or how complex its trigger is. That distinction matters for a reason beyond taxonomy. The same AI-scaled search that found a researcher's hypothesized bug across a whole subsystem in about an hour is a capability, not a permission; it does not stay confined to the defensive teams who operate it under disclosure norms, and an anti-pattern that becomes public is now searchable by anyone with a comparable tool, on the same fast timescale we observed in the copy-on-write pattern. That asymmetry is itself a reason for defenders to treat AI-scaled sweeps as an urgent, ordinary part of patching, not an optional upgrade.

This paper makes three contributions.

First, we give a primary-verified account of two 2026 Linux kernel bug families, epoll's sibling race conditions and a copy-on-write (COW) invariant broken in two kernel subsystems, cross-checked against the National Vulnerability Database (NVD), vendor advisories, and the discoverers' own writeups rather than press summaries alone (Sections III and IV).

Second, we show, from this evidence, that the human-AI discovery boundary is better described by hypothesis novelty than by the presence of a race condition: AI assistance, whether an autonomous code review or a human-directed tool, closes out instances of an already-suspected pattern quickly, while originating a novel invariant violation, or recovering from a diagnostic signal that a fix just removed, stays a human result in both families we studied (Section VI).

Third, we produce a curated table of four root-capable kernel bugs whose discovery attribution we verified against primary sources, built in part from an independently supplied list of 86 CVE identifiers that we classified in full rather than sampled, and we name every kernel bug the funnel retains, so each step from raw count to verified evidence is auditable (Section V).

From this evidence we argue for two concrete defensive investments, not one: systematic AI-scaled sweeps the moment a human characterizes a new anti-pattern, and sustained human capacity, bounty programs among them, for the bug classes that have no template yet, plus two narrower practices, re-auditing a fix's immediate neighborhood rather than trusting

pre-fix dynamic-analysis coverage to still apply, and funding the triage an inflated CVE count now demands (Section VI). Section VII states plainly what a four-row curated corpus can and cannot support, and Section VIII closes.

II. Background: The CVE Reporting Confound

Any argument about how many serious vulnerabilities appeared in the Linux kernel in 2026 has to clear one hurdle first: the kernel's own CVE count stopped tracking severity years earlier.

kernel.org became a CVE Numbering Authority (CNA) for the Linux kernel on February 13, 2024, empowered to assign CVE identifiers to kernel bugs directly rather than waiting on a third party [3], [4]. The kernel's own process documentation states its policy plainly: "the possibility of exploitation is often not evident when the bug is fixed. Because of this, the CVE assignment team is overly cautious and assign CVE numbers to any bugfix that they identify" [2]. The volume that followed shows what "overly cautious" means in practice. The kernel's CNA assigned more than 200 CVEs in its first four days, most with no demonstrated security impact [5]. The volume held through the year, by every measure taken of it: the maintainer's own year-to-date tally reached 550 identifiers reserved, assigned, or rejected by May 2, 2024 [6]; new identifiers arrived at roughly 55 a week by the autumn [7]; and 4,325 were published across 2024 in total [1]. Those three figures count different things, and that is itself the point: none of them is a severity signal, and chaining them into a single rising curve would already be an error of the kind this section warns against. We give them precisely, and cite the maintainer's own count over a rounder, unsourced figure, because precision here is unusually easy to get wrong and unusually important to get right.

The consequence is not academic. Reviewers and vendors downstream of the kernel now face a stream of identifiers where the overwhelming majority carry no demonstrated exploitability, a burden serious enough that one commentator called it "a huge problem" for organizations in regulated environments, citing hospitals as an example of a deployment where every patch requires a certification cycle [7]. A raw CVE count, in this environment, measures the kernel's own bugfix cadence and reporting policy at least as much as it measures anything about attacker or defender capability. Treating a rising CVE count as evidence that AI is making the kernel less secure, or a stable one as evidence that it is not, mistakes a policy artifact for a capability signal.

This is why the case studies in Sections III and IV, and the table in Section V, work at the level of individual, primary-verified bugs rather than counts. Each entry in Table I is root-capable by a cited technical description, not by the fact of holding a CVE identifier, and each carries a discovery attribution we verified against a primary source rather than a default assumption. That is the only unit of evidence this paper treats as meaning anything.

III. Case Study: The Epoll Sibling Bugs

A single 2023 change to the Linux kernel's epoll event-notification code introduced two separate race conditions that both surfaced as CVEs in 2026 [8], [9]. The pair gives us a natural experiment: the same code region produced one bug an AI system caught and one it did not, with a human closing the gap later.

The first bug, CVE-2026-43074, is a flaw adjacent to a use-after-free (UAF) in `ep_free()`: the kernel could free an `eventpoll` structure while another thread still held a reference to it [10]. Anthropic's Mythos model, working under the defensive research initiative Project Glasswing, found it [9], [11]. The credit is Jaeyoung Chung's own, given alongside his disclosure of the sibling bug, not the vendor's self-report. The fix landed earlier in 2026 and deferred the structure's release to a read-copy-update (RCU) grace period, a standard and conservative way to close a use-after-free window [9].

The second bug, CVE-2026-46242 and nicknamed "Bad Epoll," sat in the same code region and went unnoticed until Chung reported it to Google's kernelCTF program [9]. It is a use-after-free in `ep_remove()`: the function clears `file->f_ep` under a lock but keeps using `file` inside the critical section, racing a concurrent `__fput()` [12]. The race window is about 6 instructions wide, yet reliable: controlled tests put exploit success near 99 percent [9]. It roots ordinary Linux servers and desktops, and Android devices as well [9]. It is reachable from inside Chrome's renderer sandbox, a boundary that stops nearly every other kernel bug on a modern browser [9]. The fix, commit `a6dc643c6931`, pins the file reference before the function's critical section runs [13].

The detail that matters most for this paper is why Bad Epoll stayed hidden as long as it did. Deferring `eventpoll`'s release to an RCU grace period, the fix for CVE-2026-43074, changed the memory-reclamation timing for the exact structure Bad Epoll's race depends on, and in doing so it suppressed the Kernel Address Sanitizer (KASAN) signal that dynamic analysis would otherwise have raised in that code path [9]. Read one way, this looks like a capability gap: an AI system reviewed the code and missed a live root bug sitting a few functions away. Read more precisely, a routine, correct fix for one memory-safety bug quietly removed the diagnostic signal for its neighbor, and no one, human or AI, was set up to notice a signal that had just gone dark. That is a sharper and more useful finding than "the model failed." It says that a memory-safety fix changes what the surrounding code will report to a fuzzer or a sanitizer, and that change deserves a second look on its own, independent of whether the original fix was correct.

IV. Case Study: The Copy-on-Write Pattern

The second case study is not one bug and its sibling but a single invariant, broken in two different kernel subsystems and rediscovered independently within three weeks once it became public. The invariant is this: a buffer-sharing optimization assumes it owns a private copy of a page, when the page is in fact backed by the shared, file-backed page cache. An in-place write meant for a private buffer lands in the page

cache instead, and an unprivileged process can turn that into root. The broader pattern, an unprivileged write into read-only, file-backed page-cache memory, is not new: it drove the well-known Dirty COW (CVE-2016-5195) and Dirty Pipe (CVE-2022-0847) privilege escalations of earlier years [14], [15].

Theori researcher Taeyang Lee disclosed the first instance, CVE-2026-31431 and nicknamed "Copy Fail," on April 29, 2026 [16]. Its root cause is a 2017 optimization to the kernel's authenticated-encryption (AEAD) crypto path that puts page-cache pages in a writable scatterlist, the list of memory segments a crypto operation is allowed to write, separated from the legitimate write region by nothing more than an offset [16]. Red Hat's advisory confirms the affected range, every kernel from 4.14 onward [17]. Lee's own account of the discovery is the clearest illustration in either case study of how the two kinds of work in this paper divide: he first worked out the hypothesis himself, that the crypto subsystem's handling of page-cache-backed data was worth distrusting, and only then used Theori's AI auditing tool, Xint Code, to scale that one hypothesis into a systematic search across the whole crypto subsystem [16]. The search took about an hour, from a single prompt, once the hypothesis existed. Unlike the epoll pair, Copy Fail is not a race. It is what Lee calls a straight-line logic flaw, deterministic and reproducible on demand [16].

The same invariant surfaced again three weeks later, in a different subsystem and by the opposite method. JFrog researchers Eddy Tsalolikhin and Or Peles reported CVE-2026-43503, "DirtyClone," on May 19, after a manual audit of recent kernel patches that names no automated tool at all [18]. This time the flaw was in the networking stack's socket-buffer cloning rather than the crypto path, but the underlying technique is the same one: JFrog's writeup describes "tricking the kernel into treating read-only, file-backed page cache memory as writable network buffers" [18]. The fix merged into mainline two days later [18].

The same pattern produced further reported variants in the same three weeks, and Red Hat documents them as genuine kernel privilege-escalation bugs [19]. We leave them out of our evidence base on purpose: their discovery attribution rests on an outside firm's speculation, a single outlet's claim, or a CVE identifier we could not reconcile with the record published under it. A study of who found a bug should not build on attributions that do not hold up, so those variants appear only in the funnel count of Section V, never in Table I.

Copy Fail and DirtyClone are enough to make the point the pattern was always going to make. The same invariant fell to an AI-scaled search of one subsystem and to a fully manual audit of another, three weeks apart. What the two discoveries share is not a tool, since one used Theori's and the other used none. What they share is that the hypothesis, that in-place buffer sharing can leak into the page cache, was already in hand.

V. A Curated Evidence Table

Section II argued that a raw CVE count cannot carry a capability claim. This section shows what curation looks like in practice, applied to a concrete, independently supplied set of 86 CVE identifiers from 2026, none selected by us. Table I gives our deep evidence: the four root-capable bugs across the two families from Sections III and IV whose discovery attribution we could confirm against a primary source.

All four are root-capable: each gives an unprivileged local user full root, and Bad Epoll and Copy Fail additionally root Android and nearly every mainstream distribution since kernel 4.14, respectively. Every row carries a primary-confirmed discovery attribution, which is the point of the last column: these are the bugs whose discovery story we could pin to a primary source, a discoverer's own writeup, a vendor advisory, or a bounty-program record. The copy-on-write pattern produced further reported variants in the same three weeks, but their attribution rested on an outside firm's speculation, a single outlet's claim, or a CVE identifier we could not reconcile with its own record, so we set them aside (Section IV). Declining to tabulate a discovery we could not confirm is the same discipline the funnel below applies to the raw count: a table meant to certify who found a bug should hold only the attributions that survive checking.

Table II shows where this deep table came from. We took the 86 CVE identifiers as supplied, with no pre-filtering, and classified each by a single search: does it affect the Linux kernel, and is it root-capable. Of the 86, only 12, or 14 percent, are Linux-kernel-specific. Three of those 12 are rows in Table I: Bad Epoll, Copy Fail, and DirtyClone. The table's fourth row, CVE-2026-43074, is not drawn from the supplied list at all: it is Bad Epoll's sibling, and it reaches this paper through the epoll disclosure rather than through the funnel. The other nine kernel bugs we report by identifier and rough severity only, having not verified their discovery attribution closely enough to place them in Table I. Three of the nine are the copy-on-write variants set aside in Section IV: CVE-2026-43284 ("Dirty Frag"), CVE-2026-31635 ("DirtyDecrypt"), and CVE-2026-46300 ("Fragnesia"). The remaining six are unrelated kernel privilege-escalation or virtualization-escape bugs: CVE-2026-31402, an NFSv4.0 replay-cache heap overflow with no discoverer credited in its NVD record; CVE-2026-23111, a logic flaw in `nf_tables`; CVE-2026-43494, a double-free in the RDS subsystem nicknamed "PinTheft"; CVE-2026-46331, a missing bounds check in traffic-control packet editing nicknamed "pedit COW," whose own name points at the same anti-pattern as Section IV; CVE-2026-46333, a ptrace race condition nicknamed "ssh-keysign-pwn"; and CVE-2026-53359, a KVM shadow-MMU use-after-free nicknamed "Januscape" that escapes a guest to the host rather than a process to root. The remaining 73 entries, 85 percent of the original 86, are not Linux-kernel bugs at all: Android, network and security appliances, web platforms, AI tooling, other operating systems, and libraries, all real and often critical, but outside a kernel-focused thesis.

The point of Table II is not the 12-of-86 ratio itself, which will shift with any differently assembled list. It is the method: a raw CVE count says nothing until it has been through exactly this kind of pass, kernel or not, root-capable or not, attribution

TABLE I

THE FOUR ROOT-CAPABLE KERNEL BUGS WHOSE DISCOVERY ATTRIBUTION WE COULD CONFIRM AGAINST A PRIMARY SOURCE, SPANNING PURE-AI, PURE-HUMAN, HUMAN-PLUS-AI, AND FULLY-MANUAL DISCOVERY. REPORTED VARIANTS OF THE COPY-ON-WRITE PATTERN WHOSE ATTRIBUTION WE COULD NOT CONFIRM ARE EXCLUDED ON PURPOSE; SEE SECTION IV.

CVE	Nickname	Subsystem	Bug class	Discovery	Confidence
2026-43074	(epoll sibling)	eventpoll, `ep_free()`	UAF-adjacent	AI (Mythos)	Primary-confirmed
2026-46242	Bad Epoll	eventpoll, `ep_remove()`	Race (UAF), 6 instr.	Human (Chung)	Primary-confirmed
2026-31431	Copy Fail	AF_ALG crypto (`algif_aead`)	Logic flaw, deterministic	Human hypothesis, AI-scaled search	Primary-confirmed
2026-43503	DirtyClone	skb cloning	Logic flaw (COW)	Human, manual audit	Primary-confirmed

TABLE II

FROM 86 SUPPLIED CVE IDENTIFIERS TO THE EVIDENCE BASE ABOVE. THE FUNNEL'S MIDDLE STEP IS A FULL CLASSIFICATION PASS, NOT A SAMPLE.

Stage	Count
Raw CVE identifiers supplied	86
Linux-kernel-specific	12
in the deep table (Table I)	3
further kernel bugs, count only	9
Not Linux-kernel	73
Not resolvable in one search	1

confirmed or not. Section II's reporting-policy confound is one reason such a pass is necessary; the gap between the attributions we could confirm and the ones we could not is another.

VI. DISCUSSION: THE HYPOTHESIS BOUNDARY AND THE CASE FOR DEFENSIVE INVESTMENT

A. The boundary is hypothesis novelty, not syntax

A simpler thesis than ours would say that AI finds easy bugs and humans find hard ones, with race conditions standing in for "hard." Table I does not support that story. Copy Fail has no race at all, by its own discoverer's description, and still took nine years to surface; DirtyClone, the same invariant in a different subsystem, fell to a human running a manual patch audit with no tool credited. What actually separates the two case studies is not race versus non-race but whether a searchable hypothesis already existed.

Bad Epoll and Copy Fail sit at opposite ends of the same axis. Mythos found CVE-2026-43074 inside a defensive program that had already chosen the kernel as a surface worth reviewing; once Copy Fail was public, Xint Code found more instances of Lee's crypto-and-page-cache hypothesis across a whole subsystem in about an hour. In both cases, the AI's job was search: cover a large surface fast. This division of labor is not new: a static-analysis query encodes a known bug pattern once and then sweeps a codebase for every instance of it, as query-based variant analysis has done for years [20]. An AI-scaled search widens the surface a single such query can cover, but it does not decide what to look for. Neither case shows an AI choosing that surface for itself, and only Copy Fail records what the hypothesis was. Bad Epoll needed the opposite. No one had a reason to distrust that particular 6-instruction window until the sibling fix, by pure side effect, took away the one dynamic-analysis signal that might have flagged it, and a human closed the gap by reasoning through the race directly, without a template to search against. The pattern's own history makes the same point from a different angle: once Lee published the hypothesis that in-place buffer sharing could leak into the page cache, the same invariant was independently rediscovered in a different subsystem within three weeks, by a manual audit that used no AI at all. The tool did not matter nearly as much as whether the hypothesis was already in hand.

B. A dual-use asymmetry that argues for urgency

The same capability that let Xint Code find the bug Lee's hypothesis predicted, across a whole subsystem, in an hour is not bound to defensive use by anything intrinsic to the technology. Nor does it belong to the teams in our case studies: two years earlier, Google's Big Sleep agent had already reported what its authors called "the first public example of an AI agent finding a previously unknown exploitable memory-safety issue in widely used real-world software" [21]. That find was in a userspace database, outside this paper's kernel scope, so it grounds the dual-use premise here rather than entering Table I. Project Glasswing, Xint Code, and Big Sleep are all operated under responsible-disclosure norms, by teams that report to vendors before publishing. Guardrails of that kind live at the level of an organization's policy and a specific product's safety training, not at the level of the underlying capability, and they do not extend to an attacker running an unaligned model, a jailbroken instance, or a comparable tool built without disclosure norms attached. An anti-pattern that becomes public, as the copy-on-write invariant did after Copy Fail's disclosure, is now searchable by anyone with an equivalent tool, on the same timescale we measured: the pattern reappeared in a second subsystem within three weeks of being named, and the AI-scaled search that found the first instance took about an hour. That timescale is the argument. It means a defender who waits to sweep a codebase for a known pattern is racing an attacker who faces no comparable delay, and the norms that kept these 2026 discoveries on the defensive side of that race are a property of who did the work, not of any tool they used.

C. Two investments, not one

The hypothesis boundary and the dual-use asymmetry together point at a specific allocation of defensive effort, not a single fix.

First, once a human has characterized an anti-pattern anywhere in a codebase, a systematic AI-scaled sweep for every other instance of it is now cheap enough that skipping it is a policy choice, not a capability limitation. Lee's hour from hypothesis to confirmed bug, and the three weeks in which the same pattern reappeared in a second subsystem, both point the same way. A defender who owns the codebase can run that sweep before an attacker does, provided the organization commits to running it the moment a pattern is named, not after the next incident. A downstream vendor that only consumes the kernel is bound instead by backport lag, so for it the same urgency falls on propagating a named-pattern fix quickly, not only on sweeping for one.

Second, that sweep is not a substitute for sustained human capacity on the class of bug that has no template yet. Bad Epoll took a dedicated bounty program, a human researcher, and no prior hypothesis to point at the specific 6 instructions that mattered; nothing in either case study suggests that capacity becomes less necessary as AI-assisted sweeps get faster, since a fast sweep can only search for a pattern someone has already named. A codebase that funds only the sweep and lets kernelCTF-style bounty programs for genuinely novel bug classes lapse is optimizing for the easier half of the problem while leaving the harder half, root vulnerabilities reachable from a browser sandbox with no template to find them by, uncovered.

Two narrower practices follow. The first comes directly from Section III. A memory-safety fix can silently remove the dynamic-analysis signal that a neighboring bug depended on, as CVE-2026-43074's RCU deferral did for Bad Epoll. That gives a specific, checkable trigger: when a fix changes memory-reclamation timing, lock scope, or reference counting around a sanitizer-instrumented structure, its immediate surroundings deserve a manual or static re-audit, rather than continued reliance on the fuzzing or sanitizer coverage that ran before the fix landed. That coverage may no longer mean what it meant an hour earlier.

The second returns to Section II's reporting-policy confound, now a defender's cost and not only a researcher's inconvenience. A CNA policy that assigns a CVE to nearly every bugfix regardless of demonstrated impact, layered on top of AI-accelerated discovery of real root-capable bugs, means the scarce resource for a security team is no longer detection. It is triage: telling which of a growing stream of CVE identifiers demonstrably reaches root, which are the kind of curated, verified entry in Table I, and which are noise. Investment in that triage capacity belongs in the same budget as the sweeps and the bounty programs above.

VII. Limitations

A four-row table is a curated illustration, not a population estimate. We chose two bug families because a single 2023 commit and a single recurring anti-pattern each gave us a natural experiment with more than one outcome; a different pair of families could shift the balance of the argument, and we have not sampled 2026's kernel CVEs at random to check how representative these two are. Our four cases are also discoveries that succeeded and were attributed; we cannot see the searches that failed, so the boundary we describe is fitted to discoveries that happened, not to a sample that also holds non-discoveries.

Discovery attribution is reported only where the underlying sources support it. Several reported variants of the copy-on-write pattern rested on a single outlet, a hedged industry speculation, an ambiguous team credit line, or a CVE identifier we could not confirm against the record published under it, and we set those aside rather than tabulate them as firm claims. A different set of journalists asking different follow-up questions could sharpen or revise any of them; until then, we would rather under-claim our evidence base than pad it.

The funnel in Table II classified 86 supplied CVE identifiers by one search each, enough to separate Linux-kernel bugs from everything else, not enough to independently verify severity or attribution for the nine kernel bugs outside our evidence base. We report those nine by identifier and rough severity only, on purpose, and we do not extend the paper's discovery-attribution argument to them.

This paper covers the Linux kernel because that is where 2026 gave us a clean natural experiment; nothing here claims that the hypothesis-novelty boundary generalizes to application-layer software, to other operating systems, or to hardware, though we would expect the same logic, search speed given a template versus origination speed given none, to transfer with different specifics. Finally, the curation itself is the work of one author cross-checking primary sources; a systematic literature-review methodology with independent double-coding would strengthen the attribution judgments behind Table I beyond what a single pass can support.

VIII. Conclusion

Two Linux kernel bug families from 2026, verified against primary sources rather than press summaries, let us replace a crude claim, that AI finds easy bugs and humans find hard ones, with a sharper one: AI assistance closes out a bug once a searchable hypothesis about it exists, and origination of that hypothesis, or recovery from a diagnostic signal a fix just removed, stays a human result. Bad Epoll needed a human because no one had reason to distrust its 6-instruction window until a sibling fix silently took away the signal that might have pointed there. The copy-on-write pattern needed a hypothesis exactly once. After that, the same invariant fell in a second subsystem within three weeks, to a fully manual audit that used no AI tool at all.

That timescale is why this is a defensive paper and not a curiosity. An anti-pattern that becomes public is searchable by anyone with an equivalent tool just as fast, whether or not that party operates under the disclosure norms the teams in our case studies did. Treating an AI-scaled sweep for a newly named

pattern as routine, and sustaining human capacity for the bug classes that have no template yet, are the two investments. Re-auditing a fix's neighborhood rather than trusting coverage that fix may have quietly invalidated, and funding the triage an inflated CVE stream now demands, are the narrower practices that go with them. A CVE count inflated by reporting policy was never going to tell a defender which of these to fund. A curated, attribution-labeled table of the bugs that actually reach root is a small step toward evidence that can.

References

[1] J. Gamblin. (2025, Jan.) 2024 CVE data review. [Online]. Available: https://jerrygamblin.com/2025/01/05/2024-cve-data-review/

[2] The Linux Kernel Organization. (2026) CVE - vulnerability handling. Living document. [Online]. Available: https://docs.kernel.org/process/cve.html

[3] CVE.org. (2024, Feb.) kernel.org added as CVE numbering authority (CNA). [Online]. Available: https://www.cve.org/Media/News/item/news/2024/02/13/kernel-org-Added-as-CNA

[4] G. Kroah-Hartman. (2024, Feb.) Linux is a CNA. [Online]. Available: http://www.kroah.com/log/blog/2024/02/13/linux-is-a-cna/

[5] Amanita Security. (2024) Dear Linux kernel CNA, what have you done? [Online]. Available: https://www.amanitasecurity.com/posts/dear-linux-kernel-cna-what-have-you-done/

[6] G. Kroah-Hartman. (2024, May) 2024 CVE numbers, so far. oss-security mailing list. [Online]. Available: https://seclists.org/oss-sec/2024/q2/199

[7] heise online. (2024, Oct.) Linux: Criticism, reasons and consequences of the CVE flood in the kernel. [Online]. Available: https://www.heise.de/en/news/Linux-Criticism-reasons-and-consequences-of-the-CVE-flood-in-the-kernel-9963850.html

[8] SecurityWeek. (2026) Proof-of-concept exploit released for Linux "Bad Epoll" root access vulnerability. [Online]. Available: https://www.securityweek.com/proof-of-concept-exploit-released-for-linux-bad-epoll-root-access-vulnerability/

[9] R. Lakshmanan. (2026, Jul.) New "Bad Epoll" Linux kernel flaw lets unprivileged users gain root, hits Android. The Hacker News. [Online]. Available: https://thehackernews.com/2026/07/new-bad-epoll-linux-kernel-flaw-lets.html

[10] NVD. (2026) CVE-2026-43074 detail. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2026-43074

[11] Anthropic. (2026) Project glasswing: Securing critical software for the AI era. [Online]. Available: https://www.anthropic.com/glasswing

[12] NVD. (2026) CVE-2026-46242 detail. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2026-46242

[13] Debian Security Tracker. (2026) CVE-2026-46242. [Online]. Available: https://security-tracker.debian.org/tracker/CVE-2026-46242

[14] NVD. (2016) CVE-2016-5195 detail. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2016-5195

[15] ——. (2022) CVE-2022-0847 detail. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-0847

[16] T. Lee. (2026, Apr.) Copy fail: 732 bytes to root on Linux distributions. Theori. [Online]. Available: https://xint.io/blog/copy-fail-linux-distributions

[17] Red Hat. (2026) RHSB-2026-002: Cryptographic subsystem privilege escalation - Linux kernel (CVE-2026-31431) - copy fail. [Online]. Available: https://access.redhat.com/security/vulnerabilities/RHSB-2026-002

[18] E. Tsalolikhin and O. Peles. (2026) Dissecting and exploiting Linux LPE variant: DirtyClone (CVE-2026-43503). JFrog Security Research. [Online]. Available: https://research.jfrog.com/post/dissecting-and-exploiting-linux-lpe-variant-dirtyclone-cve-2026-43503/

[19] Red Hat. (2026) RHSB-2026-003: Dirty frag. [Online]. Available: https://access.redhat.com/security/vulnerabilities/RHSB-2026-003

[20] GitHub. (2026) CodeQL: Query code to find all variants of a vulnerability. Query-based static-analysis variant analysis. [Online]. Available: https://codeql.github.com/

[21] Big Sleep Team. (2024, Nov.) From naptime to Big Sleep: Using large language models to catch vulnerabilities in real-world code. Google Project Zero and Google DeepMind. [Online]. Available: https://projectzero.google/2024/10/from-naptime-to-big-sleep.html