The Calculator Discipline: A Taxonomy and Pre-Send Filter for AI-Assisted Vulnerability Disclosure Hallucinations

Stuart Thomas

26 May 2026

Abstract

AI assistance has made source-code review cheap, and like every productivity multiplier in the history of engineering it has therefore made being wrong cheap. The open-source security community has spent the last eighteen months noticing the result: bug-bounty intake queues drowned in plausible-sounding but fabricated vulnerability reports, with the curl project's January 2026 closure of its HackerOne programme the headline example. The conversation so far has mostly been complaint. What is missing is a taxonomy of the failure modes, a pre-send filter that catches the most mechanical of them, and honest case studies from researchers who have themselves shipped the slop. This paper supplies all three. We propose a four-class taxonomy (bug-shape fabrication, evidence fabrication, severity inflation, trivial-as-critical), present two real disclosure withdrawals and one near-miss caught before send, and describe a working pre-send tool (`hallucination_check.py`) whose four verifiers were derived from those cases. The author is one of the people who shipped the slop; the discipline described here exists because the failure happened to him. The framing throughout is that AI is a calculator: a tool that makes a careful user faster and a careless user wrong faster. The fix is not to disown the calculator; the fix is to apply calculator discipline.

Contents

Affiliation. Independent security researcher (TriageForge), Whitby, North Yorkshire, United Kingdom. Contact: `stuart.thomas@triageforge.co.uk`. ORCID: 0009-0008-4518-0064. DOI: 10.5281/zenodo.20393083.

1 1. Introduction

AI is a calculator. It is a bit high and mighty to disown it, and people are lazy — we choose the shortest path even when the shortest path runs off a cliff. That framing is the one this paper holds throughout. A calculator made arithmetic cheap; a sloppy student then produced wrong arithmetic faster. The corrective was never to ban the calculator. The corrective was a small body of discipline a maths teacher would call obvious: check the units, check the order of magnitude, check the answer makes sense in the real world. That same discipline has not yet been applied at scale to AI-assisted vulnerability disclosure.

The community has noticed the symptom. Daniel Stenberg of the curl project coined the phrase *death by a thousand slops* in a July 2025 blog post [1] and by January 2026 had ended the project's HackerOne bug-bounty programme [2], citing the unsustainable triage cost of AI-generated noise. His public retrospectives describe a sustained pattern in which AI-assisted submissions to curl have, in his observation, consistently failed to identify genuine vulnerabilities [3]. The press coverage [4, 5] follows the same arc: high-volume, plausible-looking, technically fluent reports that do not survive verification against the actual code. Stenberg's most recent posts note that the slop rate has since fallen and the quality of AI-assisted reports has risen [6] — the problem is not unsolvable, but the discipline-shaped hole in the methodology is real and worth naming.

What the existing conversation lacks is three things. First, a *taxonomy*: not all slop fails for the same reason, and the countermeasures differ by class. Second, a *pre-send filter* that researchers can run at their own keyboard before adding to the triage queue, rather than leaving the burden entirely on maintainers. Third, *honest case studies* from researchers who have themselves shipped fabricated reports and walked them back — most of the public material reads as complaints from the receiving end, with the senders silent.

This paper supplies all three. The taxonomy is four classes derived from real cases. The pre-send filter is a Python tool, `hallucination_check.py`, whose four new verifiers were written in direct response to the failures described here. The case studies are two real disclosure withdrawals to the OpenBSD project (one to Claudio Jeker on `bgpd`, one to Theo de Raadt on `OpenSMTPD`) and one near-miss to the same project's `rpki-client`, caught pre-send by a second pair of eyes.

The paper's posture is not penitent. The author is one of the people who shipped the slop. The walk-backs are described as features of the process, not as shame. The honest provenance is part of the methodology: a paper on hallucination by an author who had not himself hallucinated would be a thinner paper.

The contribution is methodological, not technical. The four failure classes are not unique to LLM-assisted work; pre-LLM researchers also misread fixed buffers as dynamic, mis-cited fuzz output, and chained-to-RCE without working PoCs. LLMs raise the throughput. Pre-send gates were always the answer; the volume has finally made them mandatory.

2 2. Case Study 1 — `bgpd rde_community.c` (withdrawn)

On 2026-05-24 the author sent the OpenBSD `bugs@` list a vulnerability report against `bgpd`'s community-attribute handling, claiming a fixed-size out-of-bounds write in `community_ext_add()` reachable from a network peer. The maintainer, Claudio Jeker, replied asking for the PoC and AFL output. On 2026-05-25 the author sent a withdrawal e-mail. The full withdrawal is preserved in the project's findings directory at `findings/OPENBGPD_community_ext_add_WITHDRAWAL.txt`; three falsehoods are admitted by name.

F1 — bug-shape fabrication. The report described a fixed `MAX_COMMUNITIES` capacity that an attacker could overflow past. The actual function — `community_ext_add()` calling `insert_community()` at `usr.sbin/bgpd/rde_community.c:226-235` — dynamically grows its community array with `reallocarray()`. There is no fixed capacity to overflow past. The constant the patch referenced does not exist in the codebase. A single read of the upstream source would have caught this; that read was never performed before sending. The AI-assisted source review had inferred a bug-shape from a pattern that fit other BSD code, and the inference had been treated as a finding.

F2 — evidence fabrication. The report claimed "22 unique SIGSEGV crashes" from an AFL run. The only AFL output the author could locate on his system after the maintainer's question showed `saved_crashes: 0` across roughly 20 million executions, and its `afl_banner` field read `imageio_harness_afl` — a fuzz run against an unrelated macOS ImageIO harness from an entirely different research thread. The "22 crashes" number had no source. Examining the actual AFL output directory before citing it — specifically, reading the `afl_banner` and `saved_crashes` fields of `fuzzer_stats` — would have caught this immediately.

F3 — artefact fabrication. The report referenced a Python network PoC, `bgp_poc.py`. The file did not exist on the author's filesystem. It had never been written. The disclosure had been drafted around the *idea* of a PoC. An `ls` would have caught this.

The withdrawal text describes the methodology gap plainly:

> Sending unverified AI-assisted analysis to bugs@ is precisely the failure mode that wastes maintainer time, and I have no excuse for doing so. The methodology lesson — independently verify every claim in the actual current source before any disclosure — is one I am applying retroactively across all my open and pending reports.

What this case shows is not that AI cannot help with source review. It shows that the calculator-discipline of *check the answer matches the real world* was simply not present in the workflow. Three of the cheapest possible checks — read the function, read the AFL banner, list the PoC file — would have caught all three falsehoods. None were performed.

3 3. Case Study 2 — `OpenSMTPD` six-claim chain (corrected)

On 2026-05-23 the author sent a longer disclosure to `security@openbsd.org`, titled *Multiple Critical Vulnerabilities in OpenSMTPD (Buffer Overflows, UAF, Info-Leaks)*, listing six claims and framing them as chainable: *"When chained, the Information Leaks bypass ASLR/RETGUARD to allow the stack buffer overflow to achieve Remote/Local Code Execution."* Theo de Raadt replied with a single pointed question — paraphrased, with respect to the privacy of `security@` correspondence: whether the author was actually claiming to have exploited the chain to gain privileges; whether execution had been achieved.

The honest answer was no. The author had not achieved execution, had not built a chain, had not produced a working PoC against any of the six. The chain framing had been written because the disclosure as a whole *read more important* with the word *chained* in it. That is the failure mode the paper is most concerned with, because unlike the `bgpd` case it does not involve any single named lie. Each individual claim contained real source observations. The fabrication was in the framing.

Per-claim verification against current OpenBSD 7.8 amd64 source (`/usr/src/usr.sbin/smtpd/`, mtime `May 21 16:11`) was performed on 2026-05-25 and is preserved at `findings/OPENSMTPD_THEO_REPLY_VERIF`. The per-claim results, in the same order as the original disclosure:

1. **`m_get_sockaddr` (mproc.c).** Real bug-shape: the destination buffer was missing a bound check; a one-line `fatalx()` guard is correct. *Inflated*: reaching the overflow requires a peer

mproc process (lka/smtp/mta) already running as the privileged `_smtpd` user. That is `PR:H` in CVSS terms, not pre-auth remote. The disclosure had implied the latter.

2. **`lka` userinfo leak (lka.c).** Real defect: `struct userinfo` on the stack is uninitialised at function entry; the `strlcpy` populator NUL-terminates but does not zero the tail; `m_add_data(p, &userinfo, sizeof(userinfo))` ships the full structure (~1 KB including a `PATH_MAX` member) over IPC, including the post-NUL stack residue. *Inflated*: the IPC channel here is `_smtpd` → `_smtpd` — same uid, same pledge, not a privsep boundary. An attacker who has already compromised one `_smtpd` process to receive the leak does not need a leak. Defense-in-depth `memset`, not the ASLR bypass the disclosure had framed it as.

3. **`smtp_reply` vsnprintf `%.*s` (smtp_session.c).** Pattern real (the `n` returned by `vsnprintf` is not clamped before later `log_info("%.*s", ..., n, buf)` calls). *Trigger fabricated*: no current format string in `smtp_reply` can produce a write length greater than the 4096-byte buffer. The fixed-string portions plus all attacker-influenceable inputs (`s->smtpname` at most 255, `s->cmd` at most 2048, heloname at most 256) sum well under 4096. The unsafe pattern should be hardened; the security claim as written had no demonstrated input. Retracted in full.

4. **`smtp_free` wait-tree UAF (smtp_session.c).** Real UAF: `smtp_free()` does not `tree_pop` the eleven `wait_*` trees keyed by `s->id`. When `io_free` fires on `IO_TIMEOUT` / `IO_DISCONNECTED` / `IO_ERROR` mid-request and the late IPC response then arrives, the relevant `tree_xpop` returns the dangling `s` pointer and the next `strlcpy` / `smtp_connected(s)` is a use-after-free. *Inflated*: crash-grade remote DoS, conditional on the admin having configured `hostnametable` (or another lka-side table) on a listener. Not RCE. The eleven-line `tree_pop` patch in the original mail is shape-correct.

5. **`crypto_decrypt_buffer` underflow (crypto.c).** *Fabricated.* The arithmetic underflow does exist (for `inlen < sizeof tag`, `inlen - sizeof tag` underflows `size_t`), but the result lands in the safe `return 0` decrypt-failure path because the surrounding comparison `outlen < inlen - sizeof tag + sizeof iv` is trivially true against the near-`SIZE_MAX` value. The dangerous `memcpy(tag, in + inlen - 16, 16)` is never reached. The disclosure had asserted exploitability without tracing the path; the trace shows safe-by-accident. Retracted in full.

6. **`queue_message_fd_r` double-close (queue_backend.c).** Real defect: on the error path after `ofp = fdopen(fdout, "w+")` succeeds, the `err:` label calls `close(fdout)` followed by `fclose(ofp)` on the same kernel fd. *Inflated*: local-only, exercised only on queue-corruption error paths that do not involve attacker input. Low-impact latent defect. The one-line `fdout = -1` fix in the original mail is correct.

The corrected reply to Theo was sent 2026-05-26 07:00 UK after a Council-of-LLMs review pass (four LLMs read the draft for tone discipline; four of four asked for the line *"I have spent the days since putting verification discipline in place"* to be cut as self-congratulatory; three of four asked for *"Sending the original mail… was a mistake"* to be cut as grovelling). The opening sentence — *"No. I did not achieve execution"* — is the only useful answer to Theo's question, and the rest of the mail is per-claim correction without padding. The reply is preserved at `findings/OPENSMTPD_THEO_REPLY_DRAFT_v2.txt`.

Summary across the six: zero are RCE, zero are chained, two were entirely fabricated (#3 and #5), and four are real but at one to three severity steps below the original framing. The disclosure had taken six unrelated observations of unsafe-looking C and assembled them into a single inflated narrative. None of the six observations was itself made up wholesale — the framing was.

4 4. Near-Miss — `rpki-client queue_add_from_cert` (caught pre-send)

The third case is the one that did not go out. On 2026-05-25 the author's triage pipeline produced a candidate against `rpki-client main.c`, function `queue_add_from_cert`: a one-byte heap out-of-bounds read on the `uri += repourisz + 1` line at `usr.sbin/rpki-client/main.c:580`, when `cert->mft` and the canonical repo URI are exactly equal-length. The cross-check agent's first-pass verdict was *NOVEL*, with a HOLD recommendation pending a triggering input. The verification record is preserved at `findings/RPKI_CANDIDATE14_NOVELTY_CHECK.md`. Two distinct failure modes appeared.

The hash hallucination. The prior-art agent (a separate process tasked with checking whether a candidate had already been reported) returned two commit hashes — `47778fc` and `573db5e` — and attributed them to a *"Frank Denis AI-assisted reports batch 2026-05-07"* against `openbsd/src`. The cross-check agent verified via direct WebFetch against `github.com/openbsd/src`. Both URLs returned HTTP 404. The author-filter view `commits?author=jedisct1` on `openbsd/src` returned *"There isn't any commit history to show here."* The hashes do not exist in the repository, and the Denis batch was a hallucination.

For the record, and to prevent defamatory misreading: **Frank Denis had no involvement in any of this.** Neither the fabricated commits nor the AI agent's attribution of them to him reflect anything Denis did, wrote, or was aware of. He is named here only because he was the person the fabrication implicated, and naming that explicitly is the only way to keep him out of the consequences of the misattribution. The class of failure is identical to F2 / F3 in case study 1: an invented artefact cited as if real, with a real person's name attached to give it weight. The catch mechanism was independent verification by a second agent that did not trust the first agent's summary and instead pulled the cited URL directly.

The cross-function invariant. Even setting the fictional prior-art aside, the *NOVEL* verdict on the candidate itself was wrong. A third read added a cross-function invariant the first two reads had missed:

- `cert.c:820-821` is a parse-time invariant: `cert->mft` is guaranteed to be `cert->repo + "/" + filename`, i.e. at least two characters longer than `cert->repo`.
- `repo.c:1226` calls `rsync_base_uri(cert->repo)` to derive the repo's canonical URI, and `rsync.c:64`'s `rsync_base_uri()` crops to `rsync://host/module`, so the derived URI is a prefix of `cert->repo` (shorter than or equal to it).
- Therefore `cert->mft` is strictly longer than (canonical repo URI) + 1, and the `uri += repourisz + 1` increment lands inside `cert->mft`'s body, never past it.

The length-equal collision the OOB-read claim required is structurally impossible because the parse-time validator in `cert.c` rejects any cert whose `cert->mft` does not have a slash followed by a filename after `cert->repo`. The candidate was a false positive. The first two reads missed it because they only examined `main.c` and did not trace `cert->mft`'s populator. The third read caught it.

The two failure modes here have different countermeasures and the paper is careful to keep them separate. The hash hallucination is mechanical and a tool can catch it (verify every cited URL resolves; verify every cited commit hash exists in the cited repo). The cross-function invariant miss is not mechanical; it requires a caller-bounds-tracing discipline that no current tool can fully automate. The latter is the limit of what a pre-send filter can do, and §6 is honest about that.

What this near-miss demonstrates is the value of a second pair of eyes. The first agent was confident. The second agent was confident. Only the third read, with the explicit prompt "trace `cert->mft` back to its populator before claiming the OOB is reachable," found the invariant. The pre-send gate is not one check; it is several, run by different parties, with the brief to disagree.

5 5. A Four-Class Taxonomy

Across the three cases, four distinct failure modes appear. The taxonomy below treats each as a *class*, names the source case, and pairs it with the catch mechanism that would have stopped it. The classes are not exhaustive — others certainly exist — but these four cover the entirety of the failures in §§2–4 and, in the author's experience, the majority of public AI-slop reports cited by Stenberg and others.

Class	Description	Source case	Catch mechanism
C1. Bug-shape fabrication	The bug pattern claimed does not exist in the code as it actually stands. Fixed buffer that is actually `reallocarray`-grown; OOB on a path the code guards; UAF on a pointer the code clears.	bgpd F1	Read the actual upstream function before drafting. Tool support: grep for `realloc/reallocarray/recallocarray` in the cited function body; if present, flag any "fixed-buffer overflow" claim for human review.
C2. Evidence fabrication	The supporting evidence (AFL run, fuzz corpus, ASAN trace, PoC script, commit hash, prior-art reference) does not exist or does not match what is cited.	bgpd F2, F3 + rpki-client Denis hashes	Verify every cited artefact resolves before citing it. Tool support: require AFL banner + exec count + timestamp triple for any "N crashes" claim; require PoC paths to resolve on disk; WebFetch every cited GitHub commit URL.
C3. Severity inflation	The bug-shape and the evidence are real, but the chain-to-RCE, pre-auth-network, or otherwise headline-grade framing is fabricated.	OpenSMTPD #1, #4	Per-claim CVSS with explicit threat-model fields. Distinguish PR:N from PR:H. Distinguish DoS from RCE. The phrase *"chained to RCE"* is reserved for cases where the chain has been demonstrated end-to-end, not merely posited.
C4. Trivial-as-critical	A real defect of negligible operational impact is framed as critical, usually by appealing to a security boundary that the trust model does not actually contain.	OpenSMTPD #2 (`_smtpd` → `_smtpd` framed as ASLR bypass)	Audience-aware severity calibration. Same-uid IPC is not a privsep boundary; trivial info-leaks within a single trust domain are hardening notes, not vulnerabilities. The maintainer's threat model, not the researcher's writing voice, sets the bar.

The classes form a rough difficulty gradient for the catch mechanism. C1 and C2 are *mechanical*: a script can verify whether `reallocarray` appears in the cited function body, whether `afl_banner` matches the cited target, whether a PoC file exists. C3 and C4 are *judgement-shaped*: the questions *"is this PR:N or PR:H?"* and *"is the lka↔mda channel a security boundary?"* require domain context that no general-purpose tool currently encodes. The paper's tool (§6) addresses C1 and C2 directly and offers structural hooks for C3 and C4 (a required `CALLER-BOUNDS ANALYSIS` section, a

required per-claim threat-model field) that move the human judgement to a place where its absence is visible.

The taxonomy is the paper's first contribution. It is offered not as the last word but as a starting vocabulary. Triage teams already track *signal* versus *noise* implicitly; making the noise classes explicit lets researchers self-check before sending.

6 6. The Tool — `hallucination_check.py`

The countermeasure is `tools/05_disclose/hallucination_check.py` (~35 KB, BSD-2-Clause). The tool predates the cases in §§2–4 — its original verifiers checked for file existence, file:line resolution, and CVE-format consistency in disclosure drafts — but four new verifiers were added on 2026-05-25 in direct response to the failures described above. The validation record is preserved at `findings/HALLUCINATION_CHECK_EXTENSION_VALIDATION.md`.

Verifier 7 — `bug_shape`. Triggered when a disclosure draft contains language matching "fixed-size buffer overflow" / "static buffer overflow" / "fixed-capacity overrun" near a named function. The verifier reads the cited function body from the attached source root and flags WRONG if `realloc`, `reallocarray`, or `recallocarray` appears in the function, on the basis that a function that grows its buffer dynamically does not have a fixed capacity to overflow past. The bgpd `community_ext_add` case is the motivating example; the verifier would have caught F1 had it existed at draft time.

Verifier 8 — `caller_bounds_gate`. Triggered when a draft cites `size_t` arithmetic and `memcpy` / `memmove`. The verifier requires the draft to contain a section heading matching `(?im)^(?:#+\s*)?(?:caller[- ]?bounds(?:\s+analysis)?|caller[- ]side\s+bounds)\b`. This is a *structural* gate, not a semantic one: it does not check whether the analysis is correct, it checks whether the author wrote one. The crypto_decrypt_buffer case (OpenSMTPD #5) is the motivating example — a complete caller-bounds analysis would have surfaced the safe-by-accident path before the claim was made.

Verifier 9 — `afl_evidence`. Triggered when a draft contains an "N crashes" / "M unique SIGSEGV" / "K hangs" phrase. The verifier requires the same paragraph to contain `afl_banner`, an exec count, and a timestamp or duration. If the banner is present but contradicts the target tokens (e.g. `imageio_harness_afl` cited against `bgpd`), the verdict is WRONG with the mismatch named. The bgpd F2 case is the motivating example; this verifier alone would have stopped the original report.

Verifier 10 — `poc_existence`. Triggered when a draft cites a PoC script path (e.g. `bgp_poc.py`, `exploit.c`, paths matching `*.py`, `*.c`, `*.pl` in the same line as "PoC" / "proof of concept"). The verifier resolves the path against the draft's directory, the absolute filesystem, and a configurable set of sibling directories. If unresolved, verdict WRONG. The bgpd F3 case is the motivating example.

Validation. The four verifiers were validated against three synthesised drafts and one known-good template. The withdrawn bgpd report was reconstructed from the withdrawal text and produced `afl_evidence=WRONG`, `poc_existence=WRONG`, and `bug_shape=UNVERIFIABLE` (which would resolve to WRONG when a real source root is attached). A `size_t`-underflow draft missing the required section produced `caller_bounds_gate=WRONG`; the same draft with the section present produced `CLEAN`. The known-good `OPENSMTPD_mproc_defense_in_depth_disclosure.txt` template — a different, properly-validated mproc patch the author intends to send separately — produced zero extension-verifier hits, i.e. no false positives.

What the tool does not catch. The `rpki-client` cross-function invariant miss (§4) is not detectable by any of the four verifiers. The candidate's draft would have contained a fully-resolved file:line reference, no fixed-buffer claim, no AFL output, no PoC path, and a passing caller-bounds analysis section *that was wrong*. The verifiers check structure, not correctness; the correctness of a

cross-function invariant trace is what the second pair of eyes is for. The tool is a necessary part of the gate, not the whole gate.

Dogfood against this paper. As a methodology self-check, `hallucination_check.py` was run against the draft of this paper itself with the project root as the source root. The tool returned a global verdict of *DIRTY*: four WRONG and two UNVERIFIABLE findings. On inspection, **every finding is a true positive by the tool's rules**: the cited bgpd source file (`usr.sbin/bgpd/rde_community.c`) is in the upstream OpenBSD tree, not in this project's local source root; the `community_ext_add()` definition cannot be located locally for the same reason; the "22 unique SIGSEGV crashes" phrase appears in the paper without the `afl_banner` field in the same paragraph; and the cited PoC paths (`bgp_poc.py`, `exploit.c`) do not resolve because the paper is *describing* failures whose artefacts never existed. The tool cannot distinguish *describing a fabricated claim* from *making the fabricated claim*. That is a real limitation, named here for honesty: a future verifier improvement would add a *quoted-context* flag (e.g., source ranges inside block quotations or marked as historical-failure exhibits) that suppresses the verdict while preserving the audit trail. The wider point survives: when the tool is run against a real disclosure draft rather than a meta-analytical paper about disclosure drafts, the same four verdicts catch the same four failure modes — and that is the use case the tool exists for.

The wider gate, as it now stands in the author's workflow:

1. AI-assisted source review produces a draft candidate.
2. The candidate is checked against the actual current upstream source by a human read (not the original AI's summary).
3. `hallucination_check.py` runs against the draft. WRONG verdicts block send.
4. A separate Council-of-LLMs pass (multiple LLMs, fresh context, brief to disagree) reviews tone and per-claim severity.
5. Only after all four pass does the disclosure go to the project's `bugs@` or `security@` address.

Step 3 is the only step the tool itself supplies. Steps 1, 2, 4, and 5 are discipline.

7 7. Discussion

The calculator analogy holds. A calculator made arithmetic cheap and let a sloppy student produce more wrong arithmetic per hour. The corrective was not to ban the calculator but to add three small habits — check the units, check the order of magnitude, check the answer matches the real world. An LLM makes source-review summaries cheap and lets a sloppy researcher produce more wrong summaries per hour. The same three habits map directly. *Check the units*: is this a `size_t`, a signed `int`, a `socklen_t`, and does the arithmetic respect that? *Check the order of magnitude*: a "22 unique SIGSEGV" claim that doesn't appear in any AFL output is two orders of magnitude away from zero; that should register. *Check the answer matches the real world*: does the function the disclosure describes actually look the way the disclosure describes it?

Why this is a methodology paper, not an AI paper. The four failure classes are not unique to AI-assisted research. Pre-LLM, careless researchers also misread `reallocarray` as fixed-buffer, also mis-cited fuzz output, also wrote *"chained to RCE"* without a chain. What LLMs change is the throughput. A researcher writing each sentence by hand was rate-limited by the cost of writing; the slop rate stayed inside the maintainer-attention envelope. A researcher dictating bullet points to an LLM and accepting the output without a verification pass can produce a finished-looking ten-claim disclosure in an hour. The pre-send gate was always the right answer to a methodology problem; the volume has just made it mandatory.

Stenberg as the canary. The curl project is the most public canary in the bug-bounty mineshaft because Stenberg writes openly about the volume he sees [1, 3, 6]. The pattern is not unique to curl. Triage teams at other open-source projects, at hosted bug-bounty platforms, and on vendor

`security@` lists report the same shape privately. The shape: high-volume, plausible-on-the-surface, technically fluent reports that do not survive a five-minute verification pass against the actual code. Closing the bug bounty, as curl did [2], is the maintainer-side defence. The researcher-side defence — pre-send filters at the keyboard — is the half of the equation this paper is concerned with. The slop will not be solved unilaterally from either side.

What we do not claim. We do not claim the four-class taxonomy is exhaustive; other classes (composability errors, misread atomicity guarantees, fabricated CVSS metric vectors) certainly exist and would benefit from naming. We do not claim `hallucination_check.py` catches everything; the rpki-client case is the explicit counter-example. We do not claim AI-assisted research is net-negative. The same research thread that produced the two withdrawals and one near-miss in this paper also produced four live OpenBSD findings disclosed during the same window, of which two (`OSPF6D-001` and `OSPFD-001`) were fixed in `-current` by Claudio Jeker on 2026-05-19 and one (`SNMPD-001`) was fixed by Martijn van Duren on 2026-05-21. AI-assisted source review found those, too. The point is not that AI is bad at vulnerability research; the point is that the discipline around it has been missing.

Limits. The tool is structural; it cannot adjudicate the correctness of a caller-bounds analysis, only its presence. The Council-of-LLMs review is only as good as the diversity of the LLMs in the council; four versions of the same model with the same training data will agree on the same mistakes. A second human pair of eyes — a peer researcher, a maintainer, a friend with the relevant subsystem knowledge — remains the strongest single check. The paper's wider claim is not that the tool replaces human judgement; the paper's claim is that the tool removes the cheapest mechanical failures so that human judgement is spent on the cases that need it.

A note on the underlying posture. What all three case studies share is a posture, not a tool: AI-assisted analysis treated as ground truth without independent verification, with the LLM's confident output allowed to substitute for the verification step. The bgpd and OpenSMTPD cases both fit. The rpki-client commit-hash hallucination fits too. The author has found it useful to give this posture a memorable internal label and to ask, before sending any disclosure, *"is this it?"* The specific label matters less than the question; the discipline is the diagnostic, not the name. No claim is made here about any particular AI tool or product — the posture is independent of the tool used.

8 8. Conclusion

AI is a calculator. The discipline of using one is older than the technology and has not changed: check the units, check the order of magnitude, check the answer matches the real world. The disclosure community has been slow to apply that discipline to AI-assisted source review, and the early-warning signal — bug-bounty intake queues drowning in slop, with curl closing its programme being the headline — is now hard to miss.

This paper has proposed a four-class taxonomy of the failure modes (bug-shape fabrication, evidence fabrication, severity inflation, trivial-as-critical), presented two real disclosure withdrawals and one near-miss caught before send, and described a working pre-send filter whose four verifiers address the two mechanical classes directly. The remaining two classes require human judgement, and the paper has been explicit that no tool currently substitutes for a second pair of eyes.

The provenance is the part the author cares about most. This paper exists because its author shipped the slop and was asked, by maintainers whose time he had wasted, what had happened. The withdrawal e-mail to Claudio Jeker on `bgpd` and the corrected reply to Theo de Raadt on `OpenSMTPD` were not academic exercises. They were apologies for failures of discipline, and the discipline described in this paper exists because those failures happened. A paper on hallucination by an author who had not himself hallucinated would be a thinner paper. The honest case studies are the work.

The invitation is open. The taxonomy, the four verifiers, and the wider workflow are released under BSD-2-Clause via the `penfold/` directory of the project's public artefacts. Bug-bounty programmes, vendor `security@` lists, and independent researchers are welcome to adopt the verifiers as-is and to extend them. Pull requests are particularly welcome on the two judgement-shaped classes (C3 severity inflation, C4 trivial-as-critical), where the current tool offers only structural hooks. The slop rate will fall when the pre-send discipline becomes routine. The case for routine pre-send discipline is the case this paper has tried to make.

9 References

[1] D. Stenberg, *Death by a thousand slops*, `daniel.haxx.se` blog, 14 July 2025. https://daniel.haxx.se/blog/2025/07/14/death-by-a-thousand-slops/

[2] D. Stenberg, *The end of the curl bug-bounty*, `daniel.haxx.se` blog, 26 January 2026. https://daniel.haxx.se/blog/2026/01/26/the-end-of-the-curl-bug-bounty/

[3] D. Stenberg, *AI slop attacks on the curl project*, `daniel.haxx.se` blog, 18 August 2025. https://daniel.haxx.se/blog/2025/08/18/ai-slop-attacks-on-the-curl-project/

[4] B. Toulas, *Curl ending bug bounty program after flood of AI slop reports*, BleepingComputer, 2026. https://www.bleepingcomputer.com/news/security/curl-ending-bug-bounty-program-after-flood-of-ai-slop-reports/

[5] *AI is drowning software maintainers in junk security reports*, Help Net Security, 18 May 2026. https://www.helpnetsecurity.com/2026/05/18/problems-with-ai-assisted-vulnerability-research/

[6] T. Krazit, *cURL's Daniel Stenberg: AI slop is DDoSing open source*, The New Stack, 2026. https://thenewstack.io/curls-daniel-stenberg-ai-is-ddosing-open-source-and-fixing-its-bugs/

[7] *AI slop got better, so now maintainers have more work*, The Register, 6 April 2026. https://www.theregister.com/software/2026/04/06/ai-slop-got-better-so-now-maintainers-have-more-work/5223172

9.0.1 Primary artefacts (project-local)

- `findings/OPENBGPD_community_ext_add_WITHDRAWAL.txt` — withdrawal e-mail to Claudio Jeker, 2026-05-25.
- `findings/OPENSMTPD_THEO_REPLY_VERIFICATION.md` — per-claim verification against current OpenBSD 7.8 amd64 source, 2026-05-25.
- `findings/OPENSMTPD_THEO_REPLY_DRAFT_v2.txt` — corrected reply to Theo de Raadt, sent 2026-05-26.
- `findings/RPKI_CANDIDATE14_NOVELTY_CHECK.md` — verification record for the `queue_add_from_cert` near-miss.
- `findings/HALLUCINATION_CHECK_EXTENSION_VALIDATION.md` — validation record for the four new `hallucination_check.py` verifiers.
- `tools/05_disclose/hallucination_check.py` — the tool itself (~35 KB, BSD-2-Clause).

10 Acknowledgements

Theo de Raadt, for the pointed reply that was the rhetorical hinge of this paper and the operational hinge of the methodology change behind it. Claudio Jeker, for the polite request for the PoC and AFL output on the `bgpd` report, which prompted the withdrawal. Martijn van Duren, for the rapid `snmpd` fix during the same disclosure window. The OpenBSD project as a whole, for the example of a security-receiving posture that takes researchers seriously enough to tell them when they are wrong. The verbatim text of `security@` correspondence is not reproduced in this paper out of respect for the list's private status; paraphrasing in §3 preserves the substance.

11 Disclosure on AI assistance

This paper was drafted with LLM assistance (Claude, Anthropic). The author independently verified every cited file path, commit hash, person's name, and URL before publication. The methodology described in this paper — read the upstream source, verify every cited artefact, structurally check the draft, take a second pair of eyes — was applied to this paper. AI assistance is used by the author as a reasonable adjustment under Equality Act 2010 §20.

12 Licence