Exploitability Evaluation Through Symbolic Reasoning: A Neuro-Symbolic Framework for Vulnerability Management

Diego Luan Monego[1], **Claudionor N. Coelho Jr.**[2], **Ricardo Dahab**[1]

[1]Instituto de Computação, Universidade Estadual de Campinas (Unicamp)
Campinas, SP – Brazil

d291074@dac.unicamp.br, rdahab@unicamp.br

[2]Majestic Labs, USA
ECE Department, Santa Clara University, USA

claudionor.coelho@alumni.stanford.edu

Abstract. *Modern vulnerability management programs face a gap between* prioritization *and* validation*: deciding which of thousands of Common Vulnerabilities and Exposures (CVEs) are actually exploitable in a specific environment. Static metrics (CVSS) and probabilistic predictors (EPSS) are insufficient; Breach-and-Attack-Simulation (BAS) tools are expensive and operationally complex. This paper presents a neuro-symbolic multi-agent framework that combines Large Language Models (LLMs) — extracting structured rules from CVE descriptions — with a Prolog engine that performs deterministic, auditable exploitability inference. On four Linux local-privilege-escalation (LPE) CVEs across vulnerable, mitigated, and patched states, the full configuration reaches 100% accuracy versus 61.5% for a single-shot LLM baseline (McNemar $p < 10^{-4}$) and 92.4% for a no-Prolog ablation, at roughly one quarter of the neural baseline's token cost. These preliminary results (four Linux LPE CVEs) suggest the approach is a promising direction — rather than a validated general solution — for a lightweight, interpretable validation layer in Continuous Threat Exposure Management (CTEM) programs.*

1. Introduction

Typical organizations remediate only $\sim$10% of the vulnerabilities present in their environments in any given window [9], making *prioritization* an essential step of vulnerability management (VM) programs. The Continuous Threat Exposure Management (CTEM) paradigm [11] explicitly separates prioritization from validation — the sub-process of deciding, given a prioritized set, which items are actually exploitable in the organization's specific operational context.

In practice, the boundary between prioritization and validation is blurry: the same instruments are often pressed into both roles, with structural limitations in either. Scores like CVSS are, by specification, expert-based assessments deliberately decoupled from real-world risk [14], and probabilistic predictors such as EPSS [7] estimate aggregate exploit likelihood — both useful for ranking, but indifferent to local configurations and therefore weak as validation signals. At the other end of the spectrum, Breach-and-Attack-Simulation (BAS) does deliver contextual evidence by emulating adversarial behavior [13], but at a cost in operational complexity that is incompatible with continuous

coverage at scale. The result is a gap: there is no deterministic, auditable, *and* cheap mechanism for contextual exploitability validation.

This paper proposes a neuro-symbolic framework that combines LLMs with a Prolog engine to fill that gap. The choice is motivated by the complementary failure modes of each paradigm: LLMs are fluent at parsing unstructured advisories and PoCs but, being probabilistic, are unreliable when a verdict requires strict deductive reasoning [1, 3]; conversely, formal solvers are deterministic and auditable but cannot, on their own, ingest the natural-language descriptions in which CVE knowledge is published. Prolog, in particular, offers a first-order Horn-clause substrate with a sound resolution procedure — a good fit for exploitability rules, which are naturally expressed as conjunctions of requirements modulated by mitigations, and whose proof tree doubles as a human-readable justification of the verdict.

In the proposed framework, LLMs extract structured rules from CVE descriptions and generate queries for host telemetry collection, while Prolog evaluates the rule against the collected facts and produces a verdict accompanied by a proof trace. We empirically validate the architecture on four Linux LPE CVEs and contrast it with two baselines: a single-shot LLM and the same pipeline with an LLM-based evaluator instead of Prolog. Preliminary results show 100% accuracy for the full configuration, with a statistically significant advantage over the neural-only baseline and a non-trivial *cost* advantage over both baselines — a property of particular relevance for fleet-wide deployment. We present this as a controlled, preliminary study, deferring cross-class and cross-platform generalization to future work.

2. Related Work

Limitations of static and probabilistic scoring. Elder et al. [5] and Spring et al. [14] document the shortcomings of CVSS for downstream decision-making — lack of empirical grounding, indifference to environmental context, oversimplification in specialized domains such as Operational Technology. EPSS [7] partially mitigates the probabilistic-ranking problem but remains opaque and indifferent to local factors (configurations, compensating controls).

BAS and operational validation. BAS platforms [4, 13] deliver high-fidelity validation by emulating adversarial behavior, but do not scale continuously due to cost and operational complexity [15]. Within the CTEM cycle [10], an open category remains: *lightweight, continuous, and explainable* validation mechanisms.

Neuro-symbolic AI in security. The integration of statistical learning with symbolic reasoning (NSAI) is described as the "third wave" of AI [6, 17]. In security, recent work applies NSAI to intrusion detection [8, 2] and privacy-preserving cybersecurity via knowledge graphs [12]. Borazjanizadeh and Piantadosi [3] and Vakharia et al. [16] (ProSLM) demonstrate the gain of coupling Prolog with LLMs for reliable reasoning. To the best of our knowledge, no prior work applies NSAI specifically to contextual exploitability assessment — the gap that motivates this proposal.

3. Proposed Architecture

We instantiate the framework on top of Velociraptor [18], an open-source endpoint visibility tool whose query language, VQL, supports on-demand collection of system artifacts

at scale and is used as the host-side execution substrate throughout the pipeline.

3.1. Overview

The architecture is organized into two flows:

1. **Rule generation (offline, human-in-the-loop):** a four-agent pipeline (collector → extractor → rule_generator → presenter) turns a CVE description into a Prolog rule declaring system- and scenario-level requirements; an analyst approves the rule before storage.
2. **Assessment (online, per host):** given an approved rule, a target host, and an attacker scenario, a five-agent pipeline (planner → collector → inferrer → evaluator → explainer) determines exploitability and produces an explanation. The *planner* (LLM) reads the `requires(CVE, system, _)` predicates and emits a VQL query [18] for each; the *collector* (Velociraptor) runs them on the target host; the *inferrer* (LLM + deterministic gating) translates results into `system_condition/2` facts; the *evaluator* (SWI-Prolog) consults the rule with the facts and answers `exploitable(CVE, Host, Scenario)`; and the *explainer* (LLM) renders the verdict and proof trace in natural language.

3.2. Prolog rule structure

Each approved rule follows a uniform schema with three predicate families. *Requirements* are declared as `requires(CVE, Kind, Atom)` facts, partitioned by `Kind` into `system` preconditions (properties of the host, e.g. `kernel_version_vulnerable`) and `scenario` preconditions (attacker capabilities granted by the engagement, e.g. `local_code_execution`) — allowing the same rule to be reused across scenarios. *Mitigations* are encoded as a many-to-many relation `mitigates(Control, Requirement)` (e.g. `mitigates(kernel_patched, kernel_version_vulnerable)`); at evaluation time the inferrer asserts `control(Host, C)` for each observed control, and the auxiliary clause `blocked(H, R) :- mitigates(C, R), control(H, C)` composes the two declaratively. *Inference* is closed by a small fixed clause set: `requirement_satisfied/2` succeeds when a requirement is observed (`system_condition/2` or `attacker_capability/2`, asserted from collected evidence) and *not* blocked, and the top-level `exploitable(CVE, Host, Scenario)` succeeds iff every system and scenario requirement is satisfied — enforced via two `forall/2` guards over the corresponding `requires/3` facts.

The split has two practical consequences: (a) the inference clauses live in a small reusable core, leaving each per-CVE rule responsible only for its requirements and mitigations; and (b) the chain from the verdict back to the satisfied/unsatisfied requirements and the mitigations involved is recovered directly from the SWI-Prolog proof trace, with no additional reporting layer.

3.3. Assertions layer: deterministic gating

We empirically observed two recurring failure modes when inference is left entirely to the LLM: (i) hallucinated version comparisons (e.g., judging kernel `5.15.0-173` as vulnerable to CVE-2023-0386, whose upstream patch lands at `5.15.0-70.77`), and

(ii) malformed VQL queries for predicates with a canonical answer (e.g., reading sysctls via file reads on procfs).

We mitigate both via a per-CVE *assertions layer* with two complementary mechanisms. *Version gates* declare, for each version-dependent predicate, the package and fixed version per (distro, release) — sourced from the corresponding Ubuntu Security Notice (USN); the inferrer compares the installed version using canonical `dpkg` semantics and suppresses the requirement fact when the host is patched, leaving an auditable comment in the Prolog program. *Predicate overrides* replace planner-generated queries for boolean predicates with a canonical command (e.g., `sysctl -n`) and apply a regex to the output, deterministically deciding whether the fact is emitted. The LLM is removed from the loop only where the answer is canonical and auditable; open-ended predicates remain its responsibility.

4. Experimental Setup

4.1. Dataset

Four Linux LPE CVEs with public PoCs, covering distinct exploit classes: heap overflow in userland (**CVE-2021-3156**, sudo Baron Samedit), environment-variable exploitation (**CVE-2021-4034**, PwnKit/polkit), kernel page-cache primitive (**CVE-2022-0847**, Dirty Pipe), and namespaces + OverlayFS (**CVE-2023-0386**). For each CVE, we maintain three Multipass snapshots: `vulnerable` (initial state), `mitigated` (configuration that neutralizes the exploit without patching — when applicable), and `patched` (fixed version installed). CVE-2022-0847 has no `mitigated` snapshot (kernel patching is the only canonical mitigation).

4.2. Conditions

We evaluate three conditions, isolating two orthogonal factors:

- **A — Full neuro-symbolic.** Pipeline as described in Section 3, with formal Prolog (SWI-Prolog) and the assertions layer enabled.
- **B — Neural-only baseline.** A single LLM call receives the CVE description (extracted from the CVE knowledge base, including PDFs), a fixed system profile (`uname`, `os-release`, `dpkg` list, SUID files, `sudoers`), and the scenario description; it emits a JSON verdict. No Prolog, no structured rule, no requirement extraction.
- **C — Symbolic-solver ablation.** Same pipeline as A through the *inferrer*; the Prolog evaluator is replaced by an LLM call prompted to derive the verdict step by step from the rule and the facts. Isolates the contribution of the formal solver.

All conditions use `gpt-4.1` as the underlying model.

4.3. Metrics

Verdict accuracy: accuracy, Cohen's κ, abstention rate, error rate, precision, recall and F1 (P/R/F1) per class (`exploitable`, `not_exploitable`), confusion matrix per condition, and the paired McNemar test (with continuity correction) for between-condition comparisons. **Cost:** mean tokens and mean latency per cell, plus an extrapolation to fleet-wide deployment.

4.4. Protocol

For every (CVE $\times$ snapshot $\times$ scenario $\times$ condition), $k = 3$ repetitions, for a total of **198 planned cells**; one cell of condition B failed transiently and was not re-executed, yielding 197 recorded outcomes ($n = 65$ for B, $n = 66$ for A and C). The k=3 repetitions re-execute the same case to probe LLM run-to-run variance, not as independent samples: the distinct situations number 22 per condition (11 CVE–state combinations $\times$ 2 scenarios; n=$66 = 22 \times 3$). We thus read the McNemar tests, paired at the run level, as evidence over this small, specific set rather than as population-level significance.

5. Preliminary Results

5.1. Accuracy by condition

Table 1. Verdict accuracy by condition. A is the full neuro-symbolic pipeline; B is the neural-only baseline; C is the no-Prolog ablation.

Cond	n	Acc	κ	Abst.	F1(expl)	F1(¬expl)
A	66	**100.0%**	**1.000**	0.0%	100.0%	100.0%
B	65	61.5%	0.163	12.3%	33.3%	76.1%
C	66	92.4%	0.781	0.0%	82.8%	95.1%

The full neuro-symbolic pipeline (A) is correct on all 66 cells. C, identical in every step except for the LLM-as-evaluator substitution, suffers 5 false positives (non-exploitable cases classified as exploitable). B exhibits high abstention (12.3%) and low F1 on the `exploitable` class (33.3%), reflecting the LLM's tendency to refuse a verdict when the fixed profile does not surface every relevant factor.

5.2. Statistical significance (paired McNemar)

Table 2. Paired McNemar tests with continuity correction. "Only X correct" counts cells where X is correct and Y is wrong (and vice versa).

X vs Y	n	Only X correct	Only Y correct	χ^2	p-value
A vs B	65	25	0	23.04	$< 10^{-4}$
A vs C	66	5	0	3.20	0.074
B vs C	65	3	23	13.89	2×10^{-4}

A vs B confirms a clear superiority of the neuro-symbolic architecture over the neural baseline. A vs C, marginally non-significant (p=0.07), indicates that the formal Prolog solver *does contribute* even when the rest of the pipeline is held constant — in all 5 disagreement pairs, A is correct and C is wrong; in none is the inverse true. Notably, this gain emerges only once the assertions layer (Section 3.3) is in place: without deterministic gating, hallucinated version comparisons upstream destabilize the facts consumed by the solver and the A-vs-C gap collapses.

5.3. Cost: per-call and at fleet scale

The full neuro-symbolic pipeline (A) is the *cheapest* of the three: the Prolog solver is a local subprocess (`swipl`), eliminating one expensive LLM call from the final stage. B

Table 3. Mean tokens and latency per cell.

Cond	Mean tokens / cell	Mean latency / cell
A	**4,025**	10.6 s
B	17,196	18.5 s
C	5,464	13.6 s

incurs the highest cost by concentrating all reasoning into a single call with long context (CVE description plus full system profile).

The token-cost gap, modest at the per-call scale, becomes decisive at the deployment scale that motivates CTEM. Continuous validation of a 10,000-host fleet against, say, 50 prioritized CVEs amounts to 500,000 cells per cycle. At public `gpt-4.1` pricing, A costs roughly $\sim$**4**$\times$ less than B and $\sim$**1.4**$\times$ less than C in tokens alone, before factoring in the wall-clock latency advantage. In other words, the symbolic component is not just a correctness lever — it is what makes continuous, fleet-wide validation **economically tractable**. These figures count only LLM tokens; telemetry collection is excluded (roughly constant across conditions) and Prolog wall-time is negligible. Rule and `assertions.yml` authoring are one-time per CVE and amortized across hosts, so the per-cell token advantage compounds with fleet size while authoring cost does not.

6. Discussion and Future Work

Implications. The results offer preliminary empirical evidence that (i) the neuro-symbolic hybrid is viable for contextual CVE validation, (ii) the contribution of the formal Prolog solver is measurable and statistically near-significant even when the rest of the pipeline is held fixed, and (iii) the system is *cheaper* in tokens than the neural-only baseline, removing a frequent barrier to operational adoption.

The assertions layer as inflection point. That A's advantage over C is contingent on the deterministic gating of Section 3.3 carries a methodological implication: NSAI evaluations in security must isolate the steps where the LLM decides questions that have a canonical, deterministic answer, lest upstream hallucination mask the architectural signal.

Limitations. (i) *Dataset size.* Four Linux LPE CVEs; generalization to RCEs, web-application vulnerabilities, or non-Ubuntu environments requires expansion. (ii) *Closed-world assumption.* Prolog treats absence of a fact as falsity, which can produce false negatives when collection is incomplete — a risk mitigated, but not eliminated, by the auditable gating comments. (iii) *Maintenance of assertions.* The per-CVE `assertions.yml` is production input analogous to a vendor advisory; updates are small per CVE but must be sustained.

Future work. Expansion of the dataset, covering additional vulnerability classes; and integration with vulnerability scanners (Qualys, Tenable) as the discovery source, with our framework as a lightweight downstream validation layer.

7. Conclusion

We presented a neuro-symbolic multi-agent framework for contextual CVE exploitability validation in CTEM. Across four Linux LPE CVEs in three snapshot states, the full architecture matches ground truth on every cell, statistically dominating a neural-only baseline

and a no-Prolog ablation at a fraction of the baseline's per-call token cost, while producing auditable proof traces. We offer this as a preliminary but promising step toward lightweight, interpretable validation complementary to heavier tools such as BAS.

Acknowledgments. We thank the anonymous reviewers for their constructive feedback, which improved the paper's framing, statistical discussion, and cost analysis.

Use of Generative AI

Anthropic Claude (Sonnet/Opus, via the Claude Code CLI) was used to assist with prose drafting and editing, refactoring of evaluation code, and formatting to the SBC LaTeX template; all factual claims and numerical results were verified by the authors. OpenAI `gpt-4.1` is an object of study, integrated into the evaluated pipeline (Sections 3 and 4.2), rather than an authoring aid.

References

[1] Coelho Jr., C. N.; Li, Y.; Tee, P. (2025). Do LLMs dream of discrete algorithms? *arXiv:2506.23408*.

[2] Bizzarri, A. et al. (2024). A synergistic approach in network intrusion detection by neurosymbolic AI.

[3] Borazjanizadeh, N.; Piantadosi, S. T. (2024). Reliable reasoning beyond natural language. *arXiv:2407.11373*.

[4] Chen, J. et al. (2023). Vulnerability correlation, multi-step attack and exploit chain in breach and attack simulation. *CloudNet 2023*.

[5] Elder, S. et al. (2024). A survey on software vulnerability exploitability assessment. *ACM Computing Surveys*, 56.

[6] d'Avila Garcez, A.; Lamb, L. C. (2023). Neurosymbolic AI: the 3rd wave. *Artificial Intelligence Review*, 56:12387–12406.

[7] Jacobs, J. et al. (2021). Exploit Prediction Scoring System (EPSS). *Digital Threats: Research and Practice*, 2.

[8] Jalaian, B.; Bastian, N. D. (2023). Neurosymbolic AI in cybersecurity: bridging pattern recognition and symbolic reasoning. *MILCOM 2023*.

[9] Kenna Security; Cyentia Institute (2019). *Prioritization to Prediction Vol. 4: Measuring what matters in remediation*. Technical report.

[10] Malevich, M. (2024). The five steps of CTEM, part 4: Validation. XM Cyber Blog. `https://xmcyber.com/blog/how-do-you-validate-security-risk/` (accessed on July 29, 2026).

[11] CTEM.org. Continuous Threat Exposure Management (CTEM). `https://ctem.org` (accessed on July 29, 2026).

[12] Piplai, A. et al. (2023). Knowledge-enhanced neurosymbolic AI for cybersecurity and privacy. *IEEE Internet Computing*, 27:43–48.

[13] Roque, M. S. C. et al. (2024). Implementation of security controls for the treatment of malware using breach and attack simulation. *INTERCON 2024*.

[14] Spring, J. et al. (2021). Time to change the CVSS? *IEEE Security and Privacy*, 19:74–78.

[15] Stein, A. (2025). Validation: the engine that powers CTEM. Cymulate Blog. `https://cymulate.com/blog/validation-the-engine-that-powers-ctem/` (accessed on July 29, 2026).

[16] Vakharia, P. et al. (2024). ProSLM: a Prolog synergized language model for explainable domain specific knowledge based question answering.

[17] Wan, Z. et al. (2024). Towards cognitive AI systems: a survey and prospective on neuro-symbolic AI.

[18] Cohen, M. (2023). Velociraptor — endpoint visibility and collection. `https://docs.velociraptor.app/`.