Ethigen: Autonomous CVE Analysis and PoC Verification With Contract-Driven Agents

1st Hugo Amaro
Instituto Pedro Nunes
Coimbra, Portugal
hamaro@ipn.pt

1st Sérgio Figueiredo
Instituto Pedro Nunes
Coimbra, Portugal
sfigueiredo@ipn.pt

3rd Pedro Conde
Ethiack
Coimbra, Portugal
conde@ethiack.com

4th Henrique Branquinho
Ethiack
Coimbra, Portugal
henrique@ethiack.com

***Abstract*—The responsible disclosure ecosystem depends on the rapid development and verification of Proof-of-Concepts (PoCs) for newly disclosed vulnerabilities. Although recent work has explored generative and agentic Artificial Intelligence (AI) for PoC generation, existing approaches are often limited to specific ecosystems and frequently suffer from context drift or brittle processing during multi-step exploitation workflows. This paper presents Ethigen, a platform for autonomous vulnerability research, PoC generation, and verification against controlled vulnerable environments. Ethigen introduces a multi-agent orchestration framework that formalizes a three-stage Research–Implementation–Verification workflow with explicit verification contracts and persistent audit trails. The system enforces a disciplined *contract–evidence* execution loop through specialized agent roles and deterministic Augments that ground agents in executable environments. To improve reliability across repeated analyses, Ethigen incrementally captures reusable execution strategies through a skill-tree mechanism, where successful workflows are distilled into structured procedural knowledge. In preliminary evaluations across 11 diverse Remote Code Execution (RCE) vulnerabilities, Ethigen successfully generated and verified 9 PoCs, against 6 success cases using Claude Code. The results show that combining machine-checkable verification harnesses with reusable procedural knowledge enables more reliable and reproducible vulnerability validation.**

***Keywords*—CVE, vulnerability, LLM, multi-agent, PoC verification**

I. Introduction

Recent advances in Large Language Model (LLM)–based agents have demonstrated strong capabilities in complex code and software engineering tasks [1], motivating their application to cybersecurity workflows. Unlike traditional automated tools, LLMs can process natural language enabling the identification and filtering of vulnerability patterns, as generate missing technical details or even exploit payloads [2] [3]. These capabilities make LLMs promising candidates for translating human-written vulnerability reports into executable Proofs-of-Concept (PoCs). This potential is particularly relevant given the inconsistent quality of vulnerability disclosures, as many Common Vulnerabilities and Exposures (CVE) reports omit critical technical details required for exploit development, forcing security researchers to reconstruct exploitation strategies through manual analysis. Leveraging learned code patterns and vulnerability semantics, LLM-based agents can partially bridge these gaps, approximating aspects of the reasoning traditionally performed by expert analysts. In contrast, conventional automated vulnerability analysis and exploitation tools rely primarily on static heuristics or rigid program analysis pipelines, which often exhibit limited coverage and low exploit success rates when applied to real-world vulnerabilities [4].

Early attempts to apply agentic AI to PoC generation have largely focused on constrained environments, such as specific package ecosystems or narrowly defined vulnerability classes [5]. However, the challenges motivating the adoption of LLM-driven approaches (e.g. incomplete vulnerability descriptions, complex exploit development workflows, or brittle automation pipelines) are pervasive across programming languages, software stacks, and deployment environments. Thus, effective autonomous vulnerability verification requires more than isolated PoC generation, demanding multi-step execution, iterative refinement, and controlled validation against realistic execution environments.

Recent research has therefore explored the integration of LLM-based generation with program analysis tools and execution feedback, enabling agents to navigate codebases, adapt exploitation strategies to runtime constraints, and iteratively refine PoCs through closed-loop validation [6] [7]. These agentic systems represent a shift from static analysis–driven security tools toward dynamic context-aware workflows. However, reliable end-to-end vulnerability verification remains challenging, particularly when tasks require coordinating multi-stage analysis, environment preparation, and execution-based validation under realistic operational conditions.

This work introduces Ethigen, an ethical detection-oriented approach for autonomous vulnerability PoC generation and verification, which differentiates itself from existing work by reframing the problem around defensive PoC synthesis and controlled empirical validation, rather than exploit construction as an end in itself. While prior focus on LLM-driven PoC generation [5] [6] [8] [9] [10] primarily evaluates exploit success rates within benchmark environments, Ethigen operationalizes a CVE-centric pipeline that standardizes heterogeneous vulnerability disclosures into reproducible evidence packages and autonomously generates runnable PoC verification scripts.

A key innovation lies in Ethigen's integration of life-cycle automation, orchestration governance, and repro-

This work was co-funded by Fundo Europeu de Desenvolvimento Regional (FEDER) through Centro2030 programme, in the scope of project nº 14357, "Ethiack Portal".

ducibility as design goals. Leveraging a multi-agent orchestration layer, the system formalizes a three-stage Research–Generation–Validation workflow with verification contracts and persistent audit trails. Unlike benchmark-centric frameworks that evaluate isolated tasks, Ethigen encodes the full CVE testing lifecycle into an auditable, project-scoped execution model. The use of containerized vulnerable scenarios, standardized outputs, and resumable batch processing supports scalable vulnerability PoC analysis while preserving traceability and controlled human intervention points. Collectively, these features advance the state of the art from experimental exploit generation towards trustworthy, operationally viable autonomous vulnerability verification systems.

Beyond orchestration, Ethigen introduces a **Skill Tree architecture** that functions as a procedural memory layer for autonomous security agents. Rather than treating each vulnerability analysis as an independent planning problem, the system incrementally captures reusable procedural knowledge derived from successful task executions. Each validated workflow is generalized into reusable **skills**, i.e., structured subplans that encode the operational structure of planning and verification operations, including vulnerability research heuristics, detection scaffolding strategies, and verification procedures. These skills are organized within a hierarchical Skill Tree and can be referenced during subsequent planning processes, allowing the system to ground new tasks in previously validated execution strategies. Transient task experience is thus transformed into persistent procedural knowledge, enabling cumulative learning across vulnerability PoC generation and verification workflows while preserving auditability through explicit skill attribution and provenance tracking.

The primary contributions of this paper are summarized as follows: (i) we introduce a novel multi-agent orchestration framework that formalizes a three-stage Research–Implementation–Verification workflow governed by explicit verification contracts and persistent audit trails; (ii) we propose a Skill Tree architecture that functions as a procedural memory layer, enabling the system to capture, generalize, and reuse validated execution strategies to improve the stability of multi-step planning; and (iii) we provide a preliminary empirical evaluation across 29 real-world remote code execution (RCE) vulnerabilities, and its comparison against Claude Code in reliability and reproducibility. The paper is organized as follows: Section II presentts related work on LLM-based vulnerability research and verification; Section III describes Ethigen solution and overall implementation approach; Section IV presents evaluation targets, associated approach and results Finally, Section V describes future work.

II. Related Work

Recent advances in large language models (LLMs) have motivated their application to offensive cybersecurity tasks, particularly in the automation of vulnerability exploitation.

Early work has focused on LLM-assisted penetration testing workflows, with limited attention to autonomous PoC generation and verification.

PentestGPT [11] represents one of the first attempts to leverage LLMs as high-level controllers for automated penetration testing of web services and applications. The LLM orchestrates the pentesting workflow by suggesting attack vectors, adapting to observed outcomes, and coordinating multi-step exploitation chains. Experimental evaluation across 13 penetration testing targets and 182 subtasks demonstrated the potential of LLMs to analyse attack surfaces and manage complex exploit workflows. Nevertheless, PentestGPT does not explicitly address systematic PoC generation nor rigorous exploit verification, and its effectiveness relies heavily on predefined tools and structured environments.

To improve realism in evaluation, CVE-Bench [8] introduced a benchmark based on real-world, critical-severity CVEs coupled with a sandboxed execution framework. Complementing this effort, LLM-CVX [12] assesses the offensive potential of LLMs by measuring their ability to translate CVE descriptions into functional exploits against web applications in production-like environments. CVE-Bench emphasizes end-to-end exploit execution and validation rather than static reasoning alone. However, results remain limited, with state-of-the-art agents exploiting only approximately 13% of evaluated vulnerabilities, highlighting the difficulty of translating vulnerability descriptions into functioning exploits under realistic constraints. Beyond web-centric vulnerabilities, CyberGym [6] introduces a large-scale benchmark based on real vulnerabilities across 188 software projects, focusing on memory-related vulnerabilities and framing tasks as PoC generation and verification via execution against pre- and post-patch versions. Despite this, the best-performing systems achieve only an 11.9% reproduction rate, underscoring the gap between LLM reasoning capabilities and the operational complexity of reliable exploit reproduction.

More recently, SEC-bench [9] introduced a fully automated benchmarking framework that explicitly evaluates LLM agents on real-world software security tasks, including PoC generation under reproducible, harnessed repository setups. SEC-bench uses an automated pipeline to construct runnable repositories with vulnerability-reproduction harnesses, enabling reliable evaluation across instances. The authors report that state-of-the-art LLM code agents achieve at most 18.0% success in PoC generation, highlighting persistent limitations even under automated, standardized evaluation conditions.

More targeted approaches have sought to narrow this gap by tightly integrating LLM reasoning with program analysis and execution feedback in closed-loop pipelines. POCGEN [5] focuses on vulnerabilities in npm packages and combines static program analysis with LLM-driven generation of exploit PoCs. The system processes vulnerability descriptions, extracts constraints, generates candidate PoCs via an LLM, and validates them through execution. Within its constrained scope, POCGEN outperforms purely analysis-driven approaches, demonstrating the effectiveness of combining LLM-based generation with tool use and iterative refinement. However, its applicability remains limited to a specific ecosystem and relatively homogeneous vulnerability patterns.

Similarly, FaultLine [10] proposes an agentic pipeline in which LLMs, guided by program analysis, simulate reasoning about vulnerable Application Programming Interfaces (APIs), construct attack payloads, and iteratively refine PoCs through execution-based validation. FaultLine emphasizes structured reasoning over API misuse patterns and integrates tool-driven feedback to improve exploit quality. While the approach achieves robust results on selected real-world bugs, it exhibits limited generalization across vulnerability classes and software ecosystems, particularly when vulnerabilities involve complex state interactions or environment-dependent behavior. These frameworks rely on iterative loops where LLMs generate and refine exploits based on execution feedback. However, these workflows typically treat each vulnerability instance independently, without retaining structured knowledge of previously successful execution strategies.

Complementary to execution-focused approaches, CVE-Genie [7] addresses the upstream challenge of semantic vulnerability interpretation and exploit generation from natural-language CVE descriptions. CVE-Genie decomposes vulnerability disclosures into structured representations (e.g., preconditions and vulnerable primitives) to synthesize exploit sketches and payloads. However, it does not consistently close the execution and verification loop, and many outputs remain near-exploit blueprints rather than executable PoCs. Vuln2Action [13] similarly translates vulnerability descriptions into sequential reproduction steps, bridging passive reports and actionable exploitation. On the defensive side, ALDExA [14] extracts characteristic ”exploit strings” from CVE data and exploit code to detect attacks in real-world traffic, achieving approximately 81% success. Despite these advances, end-to-end success rates remain low and generalization is limited. Failures often stem from insufficient modeling of execution environments and weak verification mechanisms.

Another important limitation of existing systems is the absence of persistent procedural knowledge reuse across vulnerability analyses. Most LLM-based agent frameworks treat each vulnerability as an independent task, requiring the planning process to rediscover vulnerability verification and validation strategies from scratch, leading to instability in multi-step workflows, high replanning rates, and inconsistent execution.

These limitations indicate that fully autonomous, general-purpose PoC generation and verification remains an open research problem, particularly for heterogeneous software stacks, multi-stage vulnerabilities, and incomplete or ambiguous vulnerability disclosures. Addressing this challenge requires approaches that combine semantic representations of vulnerabilities with runtime environments modeling and execution feedback, while also enabling the reuse of validated procedural knowledge across tasks.

This gap motivates Ethigen, which introduces a lifecycle-oriented verification pipeline and an Skill Tree architecture that captures reusable skills from successful executions. Applied to PoC generation and verification, this approach enables more stable and reproducible autonomous vulnerability analysis.

III. Ethigen

This section describes the conceptual and implementation views of Ethigen as an autonomous vulnerability research and validation system.

A. Conceptual view

Ethigen is an autonomous CVE-to-detection platform designed to transform vulnerability disclosures into reproducible detection artifacts through a contract-driven multi-agent workflow. Rather than treating exploit development as an isolated activity, Ethigen structures vulnerability verification as a controlled pipeline that combines vulnerability interpretation, PoC module generation, and empirical validation within reproducible execution environments. The pipeline, illustrated in Figure 1 comprises three sequential phases. The **Research** phase processes aggregated vulnerability data and produces structured artifacts describing vulnerability characteristics and potential identification strategies. The **Module Generation** phase transforms these research outputs into executable vulnerability PoC modules. Finally, the **Module Validation** phase evaluates the generated modules against controlled vulnerable environments.

At the core of the system is the **Ethigen Orchestration Layer (Ethigen-OL)**, which performs designated tasks as a sequence of steps governed by a control loop comprising three agents: **Planner, Executor and Verifier** . A *task* corresponds to a pipeline stage (Research, Module Generation, or Validation) and is executed as a sequence of *steps*, each of which may require multiple *execution attempts* (execution–verification cycles), while a *replan* denotes the regeneration of a task plan or subplan after verification failure. Each task step is associated with explicit requirements—such as permitted tools or runtime environments—and a **verification contract** specifying the evidence required for completion.

Leveraging Ethigen-OL, Ethigen coordinates the end-to-end CVE processing lifecycle, decomposing vulnerability analysis into a structured workflow composed of the Research, Module Generation and Module Validation phases described above, each representing a high-level task. The **Planner** agent translates high-level CVE objectives into hierarchical execution plans and draws knowledge from reusable skills retrieved from the skill library. The **Executor** agent performs step-level execution and tool invocation within isolated workspaces, generating intermediate artifacts such as structured vulnerability reports (Phase I) or runnable PoC detection modules (Phase II). The **Verifier** agent evaluates for each performed step whether the artifacts produced by the Executor satisfy the verification contracts defined by the Planner, producing structured failure explanations and triggering retries or corrective subplans when inconsistencies or missing evidence are detected. This contract-driven execution loop ensures that each step of the 3-phased vulnerability PoC generation and validation process is explicitly validated before progression, enabling reproducible and auditable CVE analysis runs. To improve planning stability across repeated analysis, Ethigen leverages a **Skill Curator**, which distills successful execution traces into

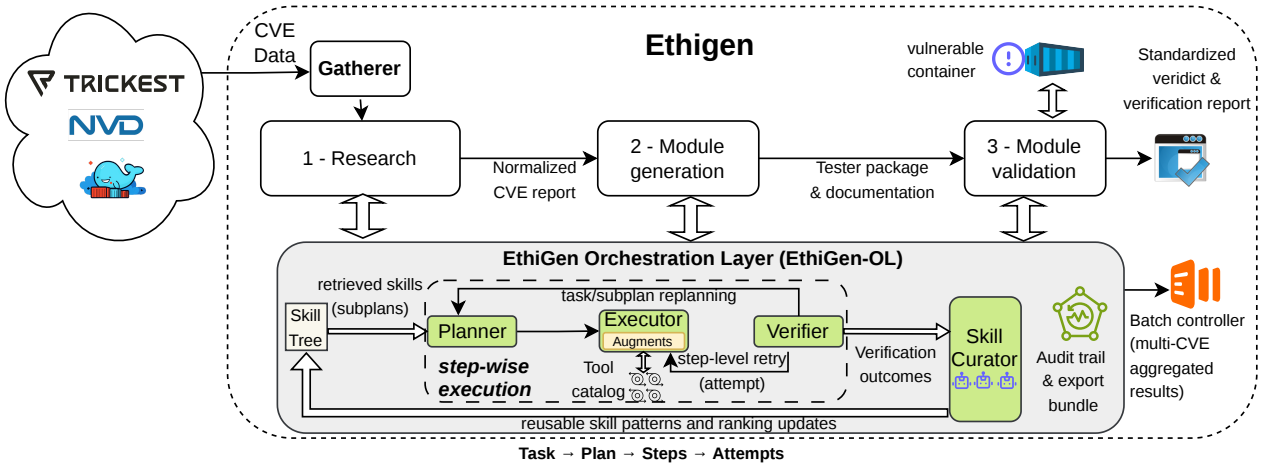

Fig. 1: Ethigen system view

reusable **skills**, i.e. structured subplans derived from prior executions. These are stored in a skill library that the Planner can reference in future tasks, enabling cumulative learning across CVE processing tasks. Finally, the **Gatherer** component feeds the orchestration layer by aggregating vulnerability intelligence from multiple sources, including vulnerability databases and public repositories, and consolidating heterogeneous CVE information into standardized input packages. Together, these components enable Ethigen to transform raw vulnerability disclosures into reproducible detection modules through a controlled and auditable workflow.

B. Implementation approach

Ethigen is implemented as a distributed backend platform built around asynchronous task orchestration and containerized execution environments. The Ethigen Orchestration Layer (Ethigen-OL) coordinates agent workflows using Celery workers and RabbitMQ message brokering. Execution state, planning artifacts, and verification results are stored in PostgreSQL, enabling persistent workflow state and reproducible task execution. The persistent model includes an event stream with detailed timestamps for each operation, ensuring execution time measurability for auditing and performance purposes.

The Gatherer component implements the ingestion pipeline responsible for aggregating vulnerability information from sources, such as the National Vulnerability Database (NVD) API and public vulnerability repositories. The aggregated data is assembled into a standardized input package for each CVE and stored in the project workspace together with containerized vulnerable environments.

The Planner, Executor, and Verifier roles described in Section II-A are implemented as specialized worker processes connected to the orchestration queue. Their interactions, depicted in Figure 1's inner gray area, are explained herein. The Planner translates each high-level objective into a hierarchical JSON subplan, optionally retrieving similar prior skills from the vector index to guide planning. Each step contains (i) a title and technical goal, (ii) detailed execution instructions, (iii) a set of required tools, such as workspace file access, patching, command execution, or Docker execution, (iv) a set of Augments, which are read-only contextual providers injected into the Executor prompt, and (v) a verification contract. The verification contract is the main interface between the Planner, Executor, and Verifier, and specifies the expected outcome of the step, acceptance criteria, optional machine-checkable checks, risk flags, and the evidence types that must be produced. The Executor materializes these steps inside isolated project workspaces provisioned through Docker containers, using Augments (e.g. real-time workspace views and constrained tool access) to reduce context drift. Each execution attempt persists validated tool parameters, result summaries, generated artifacts, and command logs in PostgreSQL and the project workspace. Upon successful task completion, the Skill Curator generalizes execution traces into reusable skills, represented as normalized plan templates or hierarchical subplans indexed by task-objective embeddings - skills construction is clarified in III-B1). During failures, the Planner applies remainder execution to generate corrective steps only for unresolved objectives, preserving validated intermediate progress. This mechanism preserves intermediate artifacts and avoids restarting the entire planning process. The overview of each Ethigen's component internal design, inputs and outputs is shown in Table I.

1) Skill representation and construction: Plan templates or Skills construction involves reconstructing the effective plan, incorporating any corrective replans or patches used during the run—and generalizing the workflow by replacing CVE-specific data (e.g., URLs, paths, filenames, or literal payloads) with abstract placeholders to extract reusable structural fingerprints. A skill in Ethigen is represented as a reusable procedural subplan rather than executable source code, en-

TABLE I: ETHIGEN COMPONENTS INPUTS, OUTPUTS AND INTERNAL DESIGN

Component	Inputs	Internal Design	Outputs
Gatherer	CVE metadata, vulnerability references, collected evidence documents, previous artifact statuses	Builds a standardized workspace per CVE; resolves artifact dependencies to determine the next artifact to produce	`input_manifest.json`, `cve.json`, evidence documents, dependency files, task objective
Planner	Task objective, workspace state, tool catalog, executor model catalog, retrieved skills, verifier replan feedback	Constructs a hierarchical JSON plan; enforces tool name validity, output declarations, loop structure, Step Contracts, and step self-containment	Plan instance with executable steps, per-step requirements, per-step contracts, and assigned executor models
Augments	Current task, step, workspace state, persistent memory, attempt metadata	Resolves read-only context queries (workspace tree, overview, persistent memory, loop state) and packages results as structured blocks	Context blocks injected into the executor prompt before each attempt
Executor	Executable step, assigned tools, assigned augments, workspace state, verifier feedback (on retry)	Runs an LLM tool-calling loop bounded by the step's tool allow-list; applies file edits, patches, shell commands, and test-environment operations as permitted by step policy	Executor message, tool call records, command outputs, modified files, workspace delta, evidence artifact references
Verifier	Per-step contract, executor output, tool call summaries, workspace delta, evidence bundle	Evaluates each contract criterion against physical evidence using read-only tools; treats executor prose as orientation only	Structured verdict (pass/retry/replan), per-criterion results, missing evidence list, root cause, recommended next action
Skill Curator	Execution traces, plan history, verifier outcomes	Abstracts CVE-specific literals from successful plans; generalizes into reusable procedural subplans; deduplicates and revises the skill library; updates per-skill success statistics	Skill patterns indexed by objective similarity and ranked by historical success rate

suring it remains agnostic to specific environments. Each skill encompasses a name, description, objective category, and execution statistics, alongside a sequence of ordered steps. Every skill step mirrors the architecture of a standard plan step, containing a title, technical goal, specific instructions, and mandatory tool and Augment requirements. Crucially, these steps include verification contracts that explicitly define the evidence and artifacts required for successful completion. These templates are stored in a relational database and indexed in a vector database via task objective embeddings. For subsequent tasks, the system queries the vector database to identify similar prior research or implementation strategies. The Planner does not blindly replay a retrieved skill; instead, it utilizes the template as a grounding reference to generate a new plan specifically adapted to the current CVE's unique constraints, the real-time workspace state, and the desired artifact objective. This approach enables the cumulative reuse of successful procedural knowledge while maintaining the architectural flexibility required for heterogeneous vulnerability research. The pseudocode for implementing a validation-related skill is depicted in Algorithm 1.

IV. Evaluation

This section presents the evaluation methodology for Ethigen and the associated results, focusing PoC generation and verification performance and the contribution of Skill Tree.

A. Evaluation plan and setup

This section presents the main evaluation goals, experiments and conditions. Two main evaluation vectors were focused:

1) Comparative performance and success rate: The primary objective was to measure Ethigen's effectiveness in autonomously generating and verifying vulnerability PoCs. The evaluation was conducted on a dataset of **29 CVEs**, with success defined as producing a *VULNERABLE* verdict confirming that the generated PoC module successfully validated the vulnerability in the target environment. The selected CVEs satisfied two criteria: (i) the availability of a corresponding **Vulhub** vulnerable environment, and (ii) coverage of multiple application categories such as web frameworks and middleware platforms. To reduce cross-class heterogeneity and ensure consistent verification signals, the evaluation focused exclusively RCE vulnerabilities, which provide clear exploitation outcomes. To contextualize Ethigen's performance against a state-of-the-art LLM-based coding agent, a subset of **11 CVEs** was also executed using Claude Code. This subset was selected due to resource and execution time constraints. Multiple LLMs were used depending on the task: GPT-5.2 was used by the Planner and the Executor for complex tasks, while GPT-5 Mini was used by the Executor for simpler tasks and by the Skill Curator. Gemini 3 Flash was used by the Executor for web-based research and summarization tasks. In the Claude Code baseline, all phases were executed using Claude Sonnet 4.5.

2) Skill Tree Ablation Study: To quantify the contribution of the Skill Tree mechanism, an ablation study was conducted by executing Ethigen with the Skill Tree enabled and disabled. The experiment was performed on a common set of 13 CVEs. For each CVE, up to three tasks corresponding to the pipeline stages - Research, Module Generation, and Module Validation - were executed, enabling a comparative analysis of success rates, replanning frequency, and contract compliance.

B. Evaluation results

The results of the experiments are presented here.

1) Comparative performance and success rate: Among the 29 CVEs, Ethigen achieved an overall success rate of

Algorithm 1 Reusable validation Skill for containerized vulnerability testing

Require: Workspace manifest, dependency files, generated tester, containerized vulnerable target
Ensure: Final verdict and evidence bundle
0: **Name:** Validate generated tester against containerized vulnerable target
0: **Description:** Run a generated tester against a prepared testbed and produce a final verdict.

0: **Step 1: Inspect validation inputs**
0: **Goal:** Identify the tester, testbed, expected outputs, and execution instructions.
0: **Tools:** `workspace_read_file`, `workspace_snapshot_tree`
0: **Augments:** `workspace_file_tree`, `holonet_memory`
0: **Contract outcome:** The tester entrypoint, testbed launch command, readiness condition, and required output files are identified.
0: **Acceptance C1:** Inputs must be grounded in the workspace manifest and dependency files.
0: **Evidence:** artifact, tool output

0: **Step 2: Start target environment and run tester**
0: **Goal:** Launch the vulnerable environment, execute the generated tester, and collect logs.
0: **Tools:** `workspace_run_command`, `workspace_read_file`, `workspace_write_file`
0: **Augments:** `workspace_file_tree`
0: **Contract outcome:** The tester has been executed against the target and execution evidence has been collected.
0: **Acceptance C1:** Logs or command output must show whether the vulnerability was triggered.
0: **Evidence:** command output, artifact

0: **return** final verdict, logs, artifacts, and evidence references =0

79% (23/29). In the common subset of **11 CVEs** used for comparison with Claude Code (Table II), Ethigen achieved **81% success (9/11)** while Claude Code achieved **54% (6/11)**. Ethigen successfully verified vulnerabilities including CVE-2023-42820, CVE-2023-51467, and CVE-2024-38856, which Claude Code failed to reproduce: these correspond to pre-authentication high or critical severity exploits affecting enterprise middleware and privileged access management systems.

2) Skill Tree Ablation Study: The ablation study shows that enabling the Skill Tree improves the robustness of the autonomous vulnerability verification pipeline. As shown in Table III[1], the system completed the full three-stage workflow for **12 of 13 CVEs** with the Skill Tree enabled, compared to **10 of 13** without it, corresponding to an increase in pipeline success rate from **76.9%** to **92.3%** (Figure 2). Stage-level analysis shows that the Research and Module Generation phases achieved near-perfect success rates in both configurations, while the most significant improvement occurs in the Module Validation phase, where generated PoC modules are executed against the target environment. Planner-level metrics (Table IV) further show that enabling the Skill Tree leads to more structured task execution, with an increase in the number of executed steps per task, as plans incorporate reusable procedural patterns. This results in a slight increase in execution attempts (i.e., repeated execution–verification cycles for individual steps), as failures are addressed through local refinement rather than plan abandonment. Consistently, the average number of failure reports per task decreases from 1.67 to 0.77, indicating fewer contract violations. In parallel, replanning frequency (i.e., regenerating a task plan or subplan after execution failure) is significantly reduced, showing that skill-guided plans are more robust. These trends are consistent with the behavior of the remainder execution mechanism, where failures are resolved through localized corrections at the step level rather than restarting the entire task. Together, these results indicate a shift from global replanning toward more stable, incremental task completion.

[1]Remaining CVEs: CVE-2023-22515, CVE-2023-46604, CVE-2023-49103, CVE-2024-21626, CVE-2024-23897, CVE-2024-27198, CVE-2024-3094, CVE-2024-3400, CVE-2024-4577

TABLE II: PERFORMANCE COMPARISON: ETHIGEN VS CLAUDE CODE

CVE ID	Ethigen	Claude Code	Source for non-success
CVE-2023-42793	S	S	N/A
CVE-2023-42820	S	F	Failed to trigger attack path.
CVE-2023-46604	S	S	N/A
CVE-2023-51467	S	F	Failed to trigger attack path.
CVE-2024-23897	S	S	N/A
CVE-2024-27348	S	S	N/A
CVE-2024-38856	S	F	Failed to trigger attack path.
CVE-2024-43441	S	S	N/A
CVE-2024-45195	F	F	Vulnerability not triggered/identified (both).
CVE-2024-4956	F	F	Both: Vulnerability not triggered/identified (both).
CVE-2024-56145	F	S	Environmental hard blockers (readiness/reachability).

[a]Success (S) indicates a 'VULNERABLE' verdict was achieved.

TABLE III: END-TO-END PIPELINE OUTCOME FOR OVERLAPPING CVES WITH AND WITHOUT SKILL TREE ENABLED. SUCCESS INDICATES THE 3 STAGES COMPLETES SUCCESSFULLY

CVE ID	With Skill Tree	Without Skill Tree
CVE-2023-42793	Success	Failure
CVE-2024-56145	Success	Failure
CVE-2024-43441	Success	Incomplete
CVE-2023-42820	Failure	Success
Other overlapping CVEs (9)	Success	Success

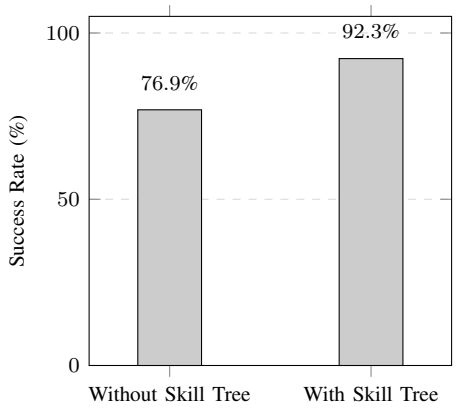

Fig. 2: End-to-end CVE pipeline success rate comparison between configurations with and without the Skill Tree

TABLE IV: TASK SUCCESS RATES AND EXECUTION BEHAVIOUR WITH AND WITHOUT SKILL TREE ENABLED

Metric	With Skill Tree	Without Skill Tree
Pipeline success by stage		
Research	13 / 13	12 / 13
Module Generation	13 / 13	13 / 13
Module Validation	12 / 13	10 / 12
Exec. and plan. behavior		
Avg. executed steps / task	3.74	2.84
Avg. execution attempts / task	4.23	3.78
Avg. replans / task	0.26	0.57

V. Conclusions and Future Work

This paper presented Ethigen, a platform for autonomous vulnerability research, PoC generation, and verification that structures vulnerability analysis into a reproducible Research–Implementation–Verification workflow. By combining multi-agent orchestration, machine-checkable verification contracts, and a skill-tree mechanism for reusable procedural knowledge, Ethigen improves the reliability of autonomous PoC generation. Preliminary evaluations across a relatively small number of RCE vulnerabilities indicate it outperforms Claude Code as a strong LLM coding agent baseline.

Future work includes the evaluation against a larger and more diverse set of CVEs, using benchmarks such as SEC-Bench, and the enhancement against potential attacks such as indirect prompt injections.

Acknowledgments

ChatGPT 5.4 and Claude Sonnet 4.6 were used to assist in textual revision (e.g. grammar correction, clarity improvement), All content was reviewed and validated by the authors.

References

[1] J. He, C. Treude, and D. Lo, "Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 5, May 2025. [Online]. Available: https://doi.org/10.1145/3712003

[2] P. Liu, J. Liu, L. Fu, K. Lu, Y. Xia, X. Zhang, W. Chen, H. Weng, S. Ji, and W. Wang, "Exploring chatgpt's capabilities on vulnerability management," in *Proceedings of the 33rd USENIX Conference on Security Symposium*, ser. SEC '24. USA: USENIX Association, 2024.

[3] A. Happe and J. Cito, "Getting pwn'd by ai: Penetration testing with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 2082–2086. [Online]. Available: https://doi.org/10.1145/3611643.3613083

[4] Z. Guo, T. Tan, S. Liu, X. Liu, W. Lai, Y. Yang, Y. Li, L. Chen, W. Dong, and Y. Zhou, "Mitigating false positive static analysis warnings: Progress, challenges, and opportunities," *IEEE Trans. Softw. Eng.*, vol. 49, no. 12, p. 5154–5188, Dec. 2023. [Online]. Available: https://doi.org/10.1109/TSE.2023.3329667

[5] D. Simsek, A. Eghbali, and M. Pradel, "Pocgen: Generating proof-of-concept exploits for vulnerabilities in npm packages," 2025. [Online]. Available: https://arxiv.org/abs/2506.04962

[6] Z. Wang, T. Shi, J. He, M. Cai, J. Zhang, and D. Song, "Cybergym: Evaluating ai agents' cybersecurity capabilities with real-world vulnerabilities at scale," 2025. [Online]. Available: https://arxiv.org/abs/2506.02548

[7] S. Ullah, P. Balasubramanian, W. Guo, A. Burnett, H. Pearce, C. Kruegel, G. Vigna, and G. Stringhini, "From cve entries to verifiable exploits: An automated multi-agent framework for reproducing cves," 2025. [Online]. Available: https://arxiv.org/abs/2509.01835

[8] Y. Zhu, A. Kellermann, D. Bowman, P. Li, A. Gupta, A. Danda, R. Fang, C. Jensen, E. Ihli, J. Benn, J. Geronimo, A. Dhir, S. Rao, K. Yu, T. Stone, and D. Kang, "Cve-bench: A benchmark for ai agents' ability to exploit real-world web application vulnerabilities," 2025. [Online]. Available: https://arxiv.org/abs/2503.17332

[9] H. Lee, Z. Zhang, H. Lu, and L. Zhang, "SEC-bench: Automated Benchmarking of LLM Agents on Real-World Software Security Tasks," in *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. [Online]. Available: https://openreview.net/forum?id=QQhQIqons0

[10] V. Nitin, B. Ray, and R. Z. Moghaddam, "Faultline: Automated proof-of-vulnerability generation using llm agents," 2025. [Online]. Available: https://arxiv.org/abs/2507.15241

[11] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "Pentestgpt: evaluating and harnessing large language models for automated penetration testing," in *Proceedings of the 33rd USENIX Conference on Security Symposium*, ser. SEC '24, USA, 2024.

[12] M. A. El yagouby, A. Lahmadi, M. Zakroum, O. Festor, and M. Ghogho, "Llm-cvx: A benchmarking framework for assessing the offensive potential of llms in exploiting cves," in *Proceedings of the 18th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '25. New York, NY, USA: Association for Computing Machinery, 2026, p. 194–205. [Online]. Available: https://doi.org/10.1145/3733799.3762978

[13] S. Kumari and G. Yadav, "Vuln2action: An llm-based framework for generating vulnerability reproduction steps and mapping exploits," *Journal of Information Security and Applications*, vol. 99, p. 104420, 2026. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214212626000505

[14] N. Ilg, M. Pfitzenmaier, D. Germek, P. Duplys, and M. Menth, "Aldexa: Automated llm-assisted detection of cve exploitation attempts in host-captured data," *IEEE Access*, vol. 13, pp. 95 379–95 391, 2025.