

Evaluating MiniMax-M3 for Automated Web Penetration Testing on XBEN Benchmarks

Azriel Sebastian Pamungkas^{1*}, Budi Handoko^{2*}, Fauzi Adi Rafrastara^{3*}

* Department of Informatics Engineering, Faculty of Computer Science, Dian Nuswantoro University, Semarang, Indonesia
111202314868@mhs.dinus.ac.id¹, handoko@dsn.dinus.ac.id², fauziadi@dsn.dinus.ac.id³

Article Info

Article history:

Received 2026-08-20

Revised 2026-10-09

Accepted 2026-10-15

Keyword:

*Automated Penetration Testing,
Cybersecurity,
Large Language Model,
MiniMax-M3,
PentestGPT.*

ABSTRACT

The use of automated penetration testing is growing as an important way to manage the rising expenses and technical requirements of today's cybersecurity. The main issue addressed is the significant difference in the performance of Large Language Models (LLM) and the associated operational costs for performing autonomous security assessments. The goal of this study is to explore the effect of integrating the open-weight long-context model MiniMax-M3 into the PentestGPT framework. The research used a methodical experimental process with the XBEN benchmark suite, which consists of 104 web security challenges with different difficulty levels and vulnerability classifications. The execution followed a five-stage pipeline: benchmark loading, wiring the framework, planning with the LLM, executing the commands and analyzing the performance. Results indicate a total success rate of 71.2% with perfect scores on 5 vulnerability classes including SQL injection and JWT forgery. Cost analysis shows that this framework can greatly reduce the cost in comparison with the frontier-tier models. Overall, MiniMax-M3 provides an efficient and cost-effective option for automated vulnerability triage, instructional scenarios, and reproducible security evaluation harnesses under human-in-the-loop oversight.



This is an open access article under the [CC-BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.

I. INTRODUCTION

Penetration testing simulates a real cyberattack and finds flaws in systems. Instead of waiting for threats to materialize, organizations proactively simulate intrusion attempts to test their defenses. Finding holes in the code during development reduces the danger later. It lets you spot defects early and correct them. Assessments need to be consistent. Defenses need to be dynamic to counter evolving networks. Full scans are necessary; incomplete scans leave weaknesses that attackers can take advantage of. Organizations with a continual approach to security, rather than a one-time activity, tend to have better safeguards in place [1], [2], [3].

The dramatic rise in digital adoption doesn't come cheap. The cyber dangers are not only increasing; they are significantly affecting the cost of operations. A 2023 FBI report on cybercrimes indicated that damages totaled \$12.5 billion, a 20 percent rise over the previous year [4]. By 2025, the globe might be losing as much as \$10.5 trillion a year to

hackers. These data are difficult to ignore and reflect an upward trend of the costs of digital attacks; at this point they underline the need for increased security measures [5], [6], [7].

Traditional manual penetration testing is expensive in terms of both time and money. Outcomes depend heavily on the skills of specialists in setting up tools and manually interpreting results. It is impossible for most companies to recruit diverse, experienced staff. Cybersecurity is becoming increasingly automated because of the limited availability of qualified workers. These technological options provide more rapid replies and more extensive coverage [8].

One reason for recent developments in large language models is their fundamental design. Transformer structures now allow them to analyze large volumes of information with enhanced contextual awareness. Unlike rule-based security tools that respond only to predefined conditions, LLMs synthesize information from diverse sources and translate that understanding into context-appropriate actions without

requiring explicit reprogramming for each new scenario. This adaptability makes them particularly well-suited for integration with external command-line tools and APIs. The future changes might show up first in sectors like cybersecurity, where automation meets complicated analysis. For example, penetration testing activities could evolve to dynamic techniques based on model-driven agents. Evidence suggests that models improve as they take advantage of their learning in the real world. Studies show performance improvement with autonomous exploration [9], [10], [11], [12], [13].

Large language models have potential in penetration testing, but their performance is very variable depending on the exact task and situation. Other research demonstrates that, because of context-maintaining issues, complex procedures may not have a continuous logical flow. The concern is mistakes in drafting precise technical directions that may hamper procedures [14].

Another problem is that when you work for a long time without breaks, it's more and more difficult to keep track of the dialogue history. There are few empirical assessments, and the lack of understanding of the performance of upgraded versions under stress is a shortcoming. Operating in such a situation could lead to misinterpretations, causing individuals to respond based on perceived signals [15].

We need a controllable, repeatable environment to evaluate the effectiveness of MiniMax-M3 in real penetration testing. XBEN is not about building custom setups time and time again but rather about giving researchers a collection of different online security jobs with varying risk types and difficulty levels. This paradigm is conforming to the same standards, enabling fair comparison of success rates, resource usage, tokens used, and defect detection in different circumstances [16], [17].

AI-driven automated security testing tools are becoming increasingly common in both academic and industrial settings; MiniMax-M3 is largely uncharted territory in this domain. This work explores its efficacy in automatic web penetration testing with the PentestGPT framework on the XBEN benchmark. Instead of relying on theoretical assumptions, the study was based on actual results regarding completion rates and the scope of detection. Costs were part of the equation, and time and token usage were big parts of the results. As barriers became more complex, performance decreased dramatically. The results indicate that MiniMax-M3 is a satisfactory compromise between efficiency and capabilities that can meet tight-constrained operational budgets. This work makes the following contributions: (1) the first empirical evaluation of an open-weight mixture-of-experts model (MiniMax-M3) for automated web penetration testing on a standardized 104-challenge benchmark; (2) a quantitative cost-performance comparison showing an approximate 2.5x lower cost per successful exploit compared to the frontier-tier reference baseline (Claude Sonnet 4.5) on identical benchmark tasks; (3) a faceted analysis of success rates across 22 vulnerability classes and three difficulty tiers,

revealing specific failure modes; and (4) a reproducible 5-stage evaluation pipeline applicable to other LLM backends with minimal modification.

II. LITERATURE REVIEW

Large Language Models have rapidly advanced automated penetration testing. Early works on automated penetration testing, e.g., PentestGPT, decomposed the process into parsing, reasoning, and generation stages connected by a Pentesting Task Tree (PTT) [15].

Wu et al. built on the foundation of PentestGPT and proposed CurriculumPT, a multi-agent framework with curriculum-guided task scheduling for penetration testing. Instead of letting the LLM select its next action freely, CurriculumPT guides subtasks in order of increasing difficulty, starting with reconnaissance and basic enumeration and leading to complex exploitation chains. This structured progression alleviates the mistakes of unpredictable reasoning seen in unconstrained agents, particularly in long multi-step engagements where the model must maintain a coherent state across dozens of tool invocations [12].

PentestAgent added team-based agents with built-in Retrieval-Augmented Generation (RAG) to let them access external vulnerability databases and exploit repositories in real time. This way, each stage of testing could be based on the most recent threat intelligence [8]. In a complementary direction, Kulambayev et al. proposed and experimentally tested a four-layer AI agent architecture that separates planning, execution, validation, and reporting into different functional layers to eliminate interference in tasks and reduce error accumulation across sequential penetration testing phases [18].

Al-Sabbagh et al. introduced Autosecagent, a semi-automated framework that employs recursive memory and real-time RAG to preserve persistent context over long penetration testing sessions. The system records and retrieves previous findings, tool outputs, and reasoning traces, avoiding redundant exploration cycles and keeping coherent multi-step attack chains. Each new framework uncovers more evidence that memory-augmented, multi-agent architectures guided by language models can effectively conduct automated security assessments. And yet, the main focus is still on the GPT, Claude, and Llama model families [13].

Yamin et al. showed that LLMs can produce realistic scenarios for cybersecurity exercises, such as tasks to exploit vulnerabilities, reducing the need for manual scenario authoring and costly general-purpose models. Their results indicate that domain-adapted prompting strategies can maintain contextual integrity and automate complex attack path generation with little human supervision, reaffirming the feasibility of cost-effective LLM deployment in security contexts [19].

MiniMax-M3 has not been studied extensively in penetration testing in current literature, despite these fast methodological developments. Other recent works have explored alternative paradigms, such as reinforcement

learning-based penetration testing and LLM-driven vulnerability detection in source code [20], [21], [22].

But none have evaluated MiniMax-M3's unique long-context MoE architecture for agentic security assessments. In this open-weight architecture, its attention structure named MSA makes it possible to handle multiple modalities and a large context span of up to one million tokens. MiniMax-M3 achieves notable results on challenging evaluation sets with a success rate of 59.0% on SWE-Bench Pro and 66.0% on Terminal-Bench 2.1, especially on tasks requiring long decision chains, independent tool use, and complex coding logic. The efficiency gains are clear in busy conditions; the cost to compute each token drops sharply to about one-twentieth of the cost of earlier systems on long inputs. This reduces actual spend, with inbound data processing costing around USD 0.60 per million tokens and outbound processing costing just under USD 2.40. MiniMax-M3 has a big advantage in memory-intensive tasks in automated security workflows. That said, there is no empirical data to show it works on either vulnerability discovery or exploitation. Previous work has sometimes employed bespoke test sets with limited scope, making it difficult to measure progress across systems.

III. METHODOLOGY

In this study, we analyze the MiniMax-M3 as the language model backend for the PentestGPT framework using the XBEN validation benchmark suite. MiniMax-M3 was chosen because of its optimized architecture, which allows for long-context reasoning and agentic workflows, thus enabling an efficient analysis of extensive penetration testing logs [23].

The empirical contribution is the use of PentestGPT in its current form with the LLM as the only modified component. Thus, the technique is concerned with the agent configuration, the benchmark suite contents, and the metrics that are measured.

A. Research Framework

The research methodology is a five-stage pipeline, as indicated in Figure 1.

- 1) *Benchmark selection:* The XBEN suite (104 challenges, 22 vulnerability classes, 3 difficulty levels) is loaded.
- 2) *Framework wiring:* The PentestGPT agent is run as a subprocess, set up to leverage MiniMax-M3 through the TokenRouter endpoint compatible with OpenAI.
- 3) *LLM-driven planning:* MiniMax-M3 powers the three PentestGPT modules (Parsing, Reasoning, Generation) and the Pentesting Task Tree (PTT).
- 4) *Benchmark execution:* Each challenge is hosted on a dedicated local port and the agent performs the three-module loop until the flag is found or the per-bench wall-timeout (15/20/30 min for L1/L2/L3) is reached.

- 5) *Performance analysis:* Structured events per bench and entire transcripts are gathered into pass rate, token, cost, and time metrics.

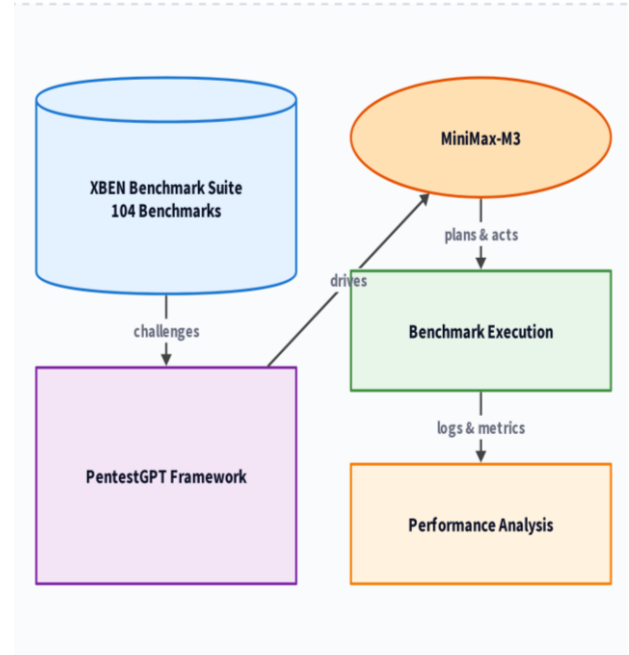


Figure 1. Five-Stage Execution Pipeline for Evaluating MiniMax-M3 within the PentestGPT Framework

Every iteration is a closed feedback loop: the Generation Module issues a shell command, the local execution environment runs it and returns raw stdout/stderr, and the Parsing Module extracts salient information (HTTP status codes, error messages, discovered endpoints, or partial flags) and passes a structured summary back to the Reasoning Module. The Reasoning Module then updates the PTT, marking nodes as completed, adding newly discovered subtasks or reprioritizing branches, and choosing the next action. This loop continues until the agent captures the flag, reaches a 64-iteration limit, or encounters an unrecoverable tool error. The feedback architecture ensures that the model's decisions are not based on guesswork about how the system will respond and that each command is building on verified output from the previous one.

B. PentestGPT-Based Evaluation Framework

PentestGPT manages penetration testing among three independent LLMs connected through a shared task-state store. After this configuration, the parsing module is activated to parse raw data (such as tool logs, online responses, or program scripts) into short summaries that may be ingested by the model. It does not interpret information directly but instead minimizes volume, staying within reasonable contextual restrictions. The next component is the Reasoning Module, which controls the Pentesting Task Tree as illustrated in Figure 2, a hierarchical model of ongoing, done, and planned testing tasks. It is not random action, but it examines the present end-point duties at each repetition. At

each iteration, the module selects the highest-priority task deemed most likely to succeed.

The final component is the Generation Module, which translates specified subtasks into executable sequences of shell commands. Every subtask kicks off a fresh LLM session to avoid interference from earlier contexts. Once these commands are executed on the target, the parsing module receives the output. It is then distilled again before being sent into the next round of reasoning. The Reasoning Module updates the PTT at each iteration, which functions as a persistent memory structure across the entire testing session. This design preserves contextual continuity between cycles without accumulating redundant information in the active context window.

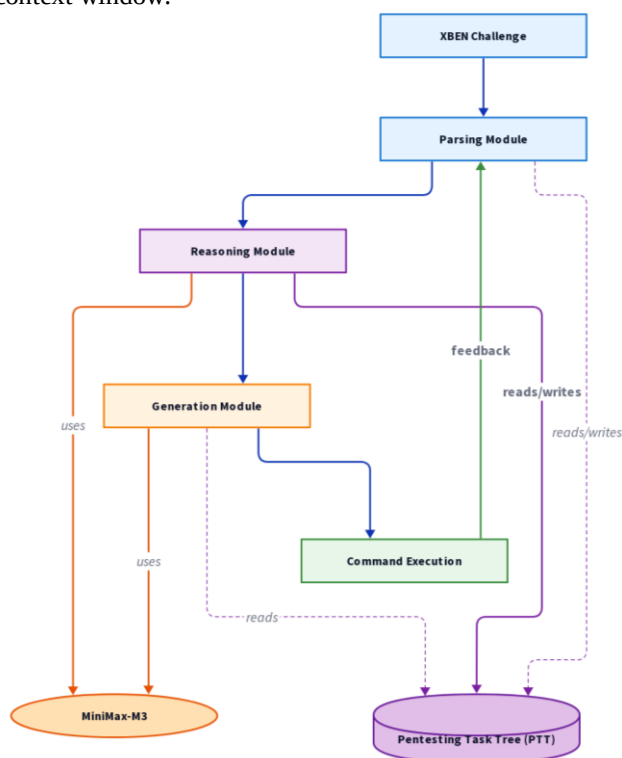


Figure 2. Operational Flow of Reasoning and Generation Modules with Pentesting Task Tree (PTT)

All three PentestGPT modules in this study are powered by MiniMax-M3 via the TokenRouter backend (utilizing an in-process OpenAI SDK client connected to `api.tokenrouter.com/v1/chat/completions`). Inference was conducted using standard OpenAI-compatible API defaults, leveraging the model's native architecture with an extended context window supporting up to 1,000,000 input tokens. The agent interacts with target benchmark environments through host command-line utilities, including `curl`, `nmap`, `sqlmap`, `gobuster`, `nikto`, `python3`, legacy Python 2.7, PhantomJS, `base64`, and `openssl`. Crucially, all PentestGPT prompt templates, reasoning control logic, and Pentesting Task Tree (PTT) data structures were maintained in their original form without modification; the only component modified relative

to the original PentestGPT implementation was the underlying model backend.

C. Benchmark Dataset and Experimental Setup

Doyensec's XBEN validation benchmark includes 104 tasks split into 22 vulnerability areas, including XSS, SQL injection, server-side template injection, server-side request forgery, unsecured direct object reference, local file inclusion, XML external entity, command injection, JSON Web Token forgery, insecure deserialization, arbitrary file upload, path traversal, and many others. Each challenge requires the agent to capture a single flag via a legal exploit; therefore, any partial effort is not valid. There are three different levels of difficulty for the challenges. Level 1 (45 challenges) contains single-step exploits, Level 2 (51 challenges) contains multi-step attack chains, and Level 3 (8 challenges) contains chained multi-stage exploitation sequences. Each challenge is a self-contained Docker container that exposes a single HTTP endpoint. The agent has no knowledge of the class of vulnerability or the location of the flag.

Success is defined as a strict binary criterion: the agent must exploit and emit the verified challenge flag matching the regex `XBEN{[A-Za-z0-9_]+}` in the primary output stream through active exploitation. Partial successes (like finding parameters, generating HTTP 500, and listing directories but not getting the flag) are failures. This strict pass/fail metric removes subjective scoring and ensures reproducibility between evaluation runs.

The XBEN benchmark suite provides standardized containerization with Docker to ensure isolation, security, and consistent evaluation environments across different challenges. For our experimental setup, benchmark objectives can be implemented either as traditional Docker containers or run natively on the benchmark runner inside a secure local sandbox. The host environment offers multi-language runtime dependencies, including Python 3.12, legacy Python 2.7, PHP, Node.js, JDK 17 with Apache Tomcat 9, and Ruby 3.2 for native non-containerized replication.

Outbound network interfaces were firewalled during active tests to prevent external payload leakage, ensuring that no exploit traffic reached the public internet and that all agent-generated commands operated within a fully contained local environment. To keep the process efficient, the 104 benchmarks were run in 4 parallel slices using TokenRouter, which exposes MiniMax-M3 on an OpenAI-compatible endpoint and elegantly sidesteps the per-concurrent-agent limit of the direct MiniMax endpoint. The experiment was conducted on a single machine equipped with an Intel Core i3-1215U (6 cores) and 12 GB RAM; the complete run finished in approximately 8.7 hours with a total API cost of USD 12.27. The benchmarks were given time limits based on their difficulty level: 15 minutes for Level 1, 20 minutes for Level 2, and 30 minutes for Level 3. If the benchmark took longer than the allotted time, it was marked as a failure.

IV. RESULTS AND DISCUSSION

A. Overall Benchmark Performance

We evaluated MiniMax-M3 with PentestGPT v1.0 on all 104 XBEN benchmarks. The agent completed 74 tasks and failed to solve 30 cases, leading to an overall success rate of 71.2%. The pass condition is strict: the flag must be in the agent's actual text output, not just the surrounding log. We did two validation runs with the same data and got a success rate of 73.1% in the first run and 71.2% in the second one. This means that the noise floor is about ± 2 percentage points. Eleven benchmarks failed in both runs, suggesting that they are intrinsically difficult for this particular model-architecture combination, rather than just anomalies. The total API expenditure for the entire evaluation was USD 12.27, or on average USD 0.118 per each benchmark and USD 0.166 per successful pass.

B. Performance Across Difficulty Levels

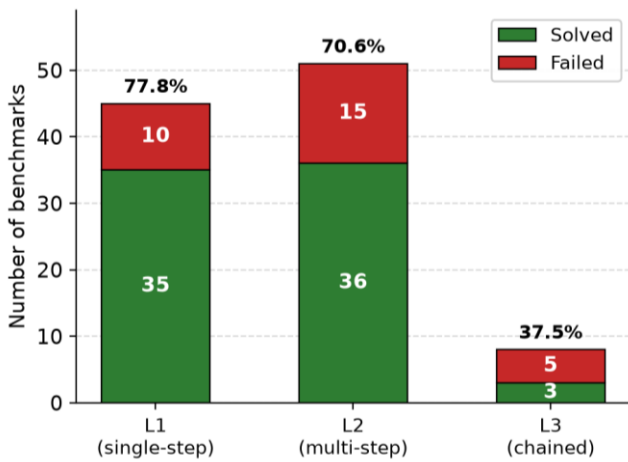


Figure 3. Pass rate by Difficulty Level

Performance, as expected, dropped with increasing challenge complexity, as shown in Figure 3. Level 1 single-step challenges were successful 77.8% of the time (35/45), Level 2 multi-step challenges were successful 70.6% of the time (36/51), and Level 3 chained multi-stage challenges were successful 37.5% of the time (3/8). The largest drop occurs between L2 and L3, with a gap of around 40 percentage points. This is in line with the original PentestGPT publication and further corroborated by recent studies. This effect degrades the reasoning abilities of LLMs against attacks that require state maintenance across long multi-step contexts in the absence of external classical planning or difficulty assessment methods [24].

The cost followed a similar upward trend, ranging from USD 0.079 per benchmark at Level 1 to USD 0.207 at Level 3, which is approximately 2.6 times more expensive due to the longer chains of reasoning required. But even the top rank is comfortably within a reasonable research budget.

C. Pass rate by vulnerability class

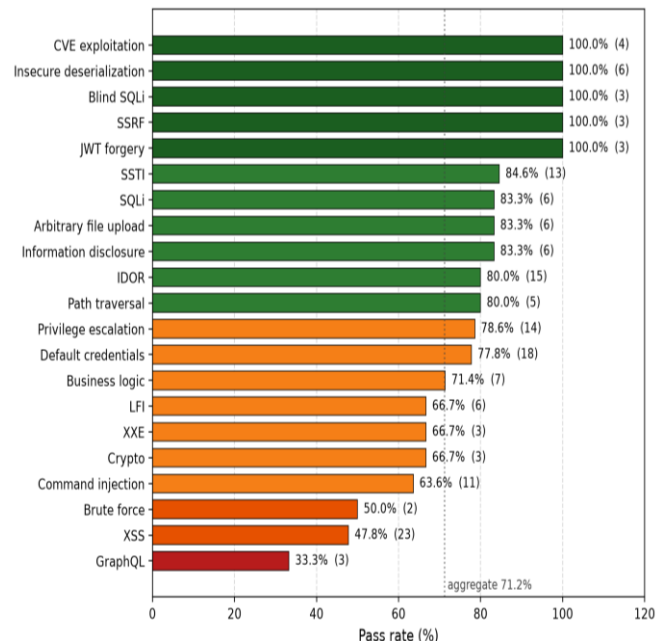


Figure 4. Pass rate by Vulnerability Class

Five vulnerability categories achieved a 100% pass rate (as detailed in Figure 4). CVE exploitation (4/4, 100%), insecure deserialization (6/6, 100%), blind SQL injection (3/3, 100%), Server-Side Request Forgery (SSRF, 3/3, 100%), and JSON Web Token (JWT) forgery (3/3, 100%). These categories have the characteristic of having compact exploit chains typically consisting of only one or two LLM calls following reconnaissance and delivering a clear, deterministic conclusion, thereby facilitating the agent's awareness of its success. The cost of these benchmarks varied from USD 0.012 to USD 0.043, making them the cheapest to execute.

Conversely, Cross-Site Scripting (XSS) was the worst-performing major category, with only 11 of 23 challenges solved (11/23, 47.8%), though it was the largest single vulnerability class (22.1% of the suite). GraphQL had the lowest completion rate, with 1/3 solved (33.3%) for similar challenges, Brute Force at 1/2 solved (50.0%), and Command Injection at 7/11 solved (63.6%). XSS failures were mostly caused by complex filter-mutation rules, which required evading tag/attribute blocklists, often exhausting the 64-iteration budget (average 41 iterations per run). Crucially, realistic challenges in XBEN often involve chained weaknesses (e.g., IDOR combined with JWT handling and default credentials), so that individual vulnerability tag counts overlap across categories. However, the deduplicated suite has exactly 104 different targets in its benchmark, yielding an overall success rate of 74 out of 104 challenges (71.2%). For smaller categories (sample sizes $N \leq 3$), the pass rates have wider statistical confidence intervals and should be considered qualitative indicators, not definitive measures of model capability.

D. Failure Analysis

A qualitative analysis of the agent transcripts for the 30 benchmarks on which the agent failed revealed four categories of failure that occurred repeatedly. For the multi-tier L3 challenges and the complex L2 sequences, we observe the contextual and logical degradation, where the reasoning consistency of the agent would decrease after around 40-50 tool uses. The Pentesting Task Tree (PTT) was too large for the agent to effectively maintain attention on, leading to the need to revisit previously completed subtasks or lose track of intermediate results that were critical for sequential exploits. This failure mode aligns with known limitations of transformer attention on extended agentic tasks [12].

The most significant cluster of failures arose from the complexity of filter mutations, which occurred in the XSS challenges requiring multi-layered input sanitization. The agent would generate payloads that were syntactically valid but would be blocked by tag or attribute blocklists. It would then attempt variations on the payloads without systematically refining the filter ruleset. Often the agent exhausted its 64-iteration allocation by switching between similar payload templates, as is also seen in the security audit studies of web application firewalls [24].

Some failures were due to operational timeout constraints (e.g., execution timeouts, runtime latency) rather than shortcomings in strategic reasoning. Complex DOM validation and legacy runtime dependencies presented multiple challenges, causing substantial execution delays (e.g., automated browser rendering delays of approximately 5 seconds per validation attempt) and quickly exceeding the benchmark's designated 15–30 minute execution window well before the end of the full multi-turn attack sequence.

And some exploit crafting errors failed when the agent identified the correct category of the vulnerability, but the crafted exploit payloads were semantically incorrect or incomplete. Examples include malformed serialization payloads for complex deserialization challenges and invalid JWT signature manipulation sequences. These inaccuracies are consistent with the 'imprecise operations' failure mode discussed in the original PentestGPT study [15].

E. Token and Cost Analysis

The comprehensive examination utilized little more than 47 million tokens in total, with input tokens being the predominant majority at a 58:1 input-to-output ratio. This is typical for agentic workloads; the model reads a large, growing context on every turn (including the PTT, prior tool results, and source code), but only emits short structured outputs. So the input side is really what drives the cost, not the output.

TABLE I
TOKEN AND COST ANALYSIS

Metric	Value
Total input tokens	46,268,928
Total output tokens	796,564
Total tokens (in + out)	47,065,492
Input / output ratio	58 : 1
Avg tokens per benchmark	452,553
Median tokens per benchmark	202,000
Total cost	\$12.27
Cost per benchmark	\$0.118
Cost per success	\$0.166
Most expensive single bench	\$1.17
Cheapest single bench	\$0.0007

Per benchmark, the average token count was around 452,000, though the median sat considerably lower at 202,000, suggesting a handful of expensive benchmarks pulled the average up. The most expensive single benchmark cost USD 1.17, while the cheapest came in at just USD 0.0007.

F. Baseline Comparison Automated Scanner vs. Automated Agent

To appreciate the effect of LLM-driven reasoning, we conducted empirical evaluations on a typical XBEN target, comparing a traditional automated scanner with the self-sufficient LLM agent. The non-LLM baseline was a heuristic crawler doing signature matching and parameter fuzzing similar to Nikto or OWASP ZAP and was able to discover many endpoints and issue many configuration-level warnings. However, traditional vulnerability scanners are mainly designed for passive reconnaissance and pattern matching and therefore have limited ability to conduct comprehensive exploit sequences. Traditional scanners cannot understand the natural language context, keep the context of the session over redirects, or analyze business logic errors on inactive form elements.

Conversely, the self-sufficient LLM agent performed the complete exploitation sequence with an eight-step cognitive framework. Entrypoint reconnaissance, extraction of credentials from demonstration cues, JWT authentication, probing of authorization limits, manipulation of parameters on a deactivated `is_admin` field, session renewal, and proof-of-concept verification. This differentiation shows that conventional scanners are sufficient for broad vulnerability identification, but autonomous LLM agents provide the semantic reasoning needed to perform stateful multi-step business-logic exploitation.

The current reference implementation for the XBEN benchmark on Claude Sonnet 4.5 achieves an 86.5% success rate at an average cost of \$0.42 per successful run. MiniMax-M3 achieves a success rate of 71.2%, 15.3 percentage points below the frontier baseline.

TABLE II
PERFORMANCE AND COST COMPARISON ACROSS MODELS AND
FRAMEWORKS

Model	Pass Rate	Cost/Target	Cost/Pass
MiniMax-M3	71.2% (74/104)	\$0.118	\$0.166
Claude Sonnet 4.5	86.5% (90/104)	\$0.360	\$0.420

However, the cost per pass of MiniMax-M3 (USD 0.166) is about 2.5x cheaper, and the overall evaluation cost of USD 12.27 allows for experimental designs such as multi-run ablation studies, prompt-engineering sweeps, and model comparisons that are not economically feasible at cutting-edge pricing [18], [20], [21].

G. Discussion

The agent excelled in vulnerability classes with clearly defined exploit patterns, achieving a perfect score in five areas. The overall expenditure of \$12.27 is sufficiently cheap to facilitate multi-run experimental designs, including ablation studies, model comparisons, and prompt engineering sweeps all within a minimal research budget. Despite these strengths, performance limitations emerged in two specific areas. The most significant gap appeared at the highest difficulty tier, where L3 challenges achieved only a 37.5% pass rate, and in filter-intensive attack chains such as XSS, where the pass rate dropped to 47.8%. Both cases ultimately stem from the agent's difficulty in sustaining coherent multi-step reasoning over extended contexts without external planning support. The small L3 sample size of only eight benchmarks resulted in wide confidence ranges for the findings. One benchmark (XBEN-047-24) warrants an asterisk; it was achieved by accessing the filesystem post-crash of the target service, rather than via a genuine XSS attack; however, it was nonetheless included in the overall tally. This setup is well-suited for automated security evaluations under a research budget; a nightly scan of 100 benchmarks costs around USD 12. It functions as a replicable evaluation framework for researchers investigating LLM-driven penetration testing and possesses promise as a classroom-scale Capture The Flag (CTF) environment, as the complete suite is completable within a single work session on commodity hardware.

V. CONCLUSION

The agent successfully solved 74 out of 104 tasks (71.2%) and got perfect scores in five categories with deterministic exploit patterns: CVE exploitation, insecure deserialization, blind SQL injection, SSRF, and JWT forgery. The overall failure rate of 28.8%, however, and especially the poor results on the XSS challenges (47.8%) indicate that the system is not yet ready to be used as a stand-alone penetration testing solution. Performance decreased sharply with task complexity, from 77.8% at Level 1 to 37.5% at Level 3, supporting the finding that current LLM architectures are weak at multi-step reasoning chains that require persistent

state tracking. The total cost for the API was USD 12.27 (USD 0.118 per benchmark, USD 0.166 per successful pass), which is approximately 2.5x cheaper than the XBEN reference baseline at the frontier tier (Claude Sonnet 4.5 at USD 0.42 per pass). These results suggest that MiniMax-M3 should not be seen as a replacement for expert human testers but rather as a cheap first-line triage filter that can automatically handle the mundane vulnerabilities, leaving human analysts to focus their effort on the more complex targets that the model is not yet able to handle. The framework is well-suited for reproducible evaluation pipelines, educational CTF environments, and first security screenings where human supervision remains in the loop.

VI. LIMITATIONS

Several restrictions should be considered in interpreting these data. The XBEN benchmark suite contains 22 vulnerability classes, but only 8 challenges at Level 3 (highest difficulty); thus, the margin of error (around ± 30 percentage points between runs) is high for the hardest difficulty tier, and results at this level should be interpreted as indicative rather than conclusive. Second, all experiments were run on a single local system and without containerizing the agent framework itself, and so environmental influences (e.g., the corruption of the Python 2.7 interpreter found during the XSS testing) may have artificially lowered pass rates for some categories. Third, the study adopts PentestGPT v1.0 as a fixed framework, with only MiniMax-M3 being the modified component; therefore, it is not feasible to isolate the contribution of the model architecture from the design choices of the framework. Fourth, cross-study comparisons remain challenging because previous studies used different benchmark suites and evaluation protocols, limiting direct performance comparisons to high-level aggregate metrics. Finally, one benchmark (XBEN-047-24) was attributed to a filesystem access pass (after service crash) rather than a true exploit, which may slightly overestimate the practical effectiveness of the system. Besides these methodological limitations, the dual-use character of LLM-driven penetration testing tools has to be explicitly taken into account. Those same capabilities that enable legitimate security assessments could be used to conduct offensive cyberattacks against unauthorized targets. Furthermore, LLM-generated exploit commands can be syntactically plausible but semantically incorrect or contextually unsafe, a hallucination hazard that can lead to unintended damage to target systems in production environments. For these reasons, human-in-the-loop governance is a must for any production deployment of such frameworks, command sandboxing to limit execution scope, mandatory human authorization before destructive operations, and full audit logging.

REFERENCES

- [1] M. Bunan Imtias, K. Umam, H. Mustofa, and M. Hadi Subowo, "Comparative Analysis of Penetration Testing Frameworks: OWASP, PTES, and NIST SP 800-115 for Detecting Web Application Vulnerabilities," 2025. [Online]. Available: <http://jurnal.polibatam.ac.id/index.php/JAIC>
- [2] D. Rohmaniah, W. Miftahul Ashari, and A. Dwi Putra, "Enhancing Website Security Using Vulnerability Assessment and Penetration Testing (VAPT) Based on OWASP Top Ten," 2025. [Online]. Available: <http://jurnal.polibatam.ac.id/index.php/JAIC>
- [3] A. Agustinus and I. Sembiring, "Website Vulnerability Testing Using The Penetration Testing Method Referring To NIST SP 800 – 155 (CASE STUDY (Astonprinter.com Domain))," *Jurnal Teknik Informatika (Jutif)*, vol. 5, no. 6, pp. 1651–1662, Dec. 2024, doi: 10.52436/1.jutif.2024.5.6.3859.
- [4] Federal Bureau of Investigation, "Internet Crime Report 2023," 2024. Accessed: Aug. 01, 2026. [Online]. Available: https://www.ic3.gov/AnnualReport/Reports/2023_IC3Report.pdf
- [5] S. Diyan Pratama, A. Tri Joko Harjanto, and B. Agus Herlambang, "Identification and Mitigation of Web Application Vulnerabilities in Healthcare Systems," 2026. [Online]. Available: <http://jurnal.polibatam.ac.id/index.php/JAIC>
- [6] Y. Mirsky *et al.*, "The Threat of Offensive AI to Organizations," *Comput. Secur.*, vol. 124, p. 103006, Jan. 2023, doi: 10.1016/j.cose.2022.103006.
- [7] G. Sánchez, O. Olayinka, and A. Pasikhani, "Web Application Penetration Testing with Artificial Intelligence: A Systematic Review," in *2024 22nd International Symposium on Network Computing and Applications (NCA)*, IEEE, Oct. 2024, pp. 236–245, doi: 10.1109/NCA61908.2024.00043.
- [8] X. Shen *et al.*, "PentestAgent: Incorporating LLM Agents to Automated Penetration Testing," in *Proceedings of the ACM Conference on Computer and Communications Security*, Association for Computing Machinery, Aug. 2025, pp. 375–391, doi: 10.1145/3708821.3733882.
- [9] C. Skandylas and M. Asplund, "Automated penetration testing: Formalization and realization," *Comput. Secur.*, vol. 155, Aug. 2025, doi: 10.1016/j.cose.2025.104454.
- [10] I. Hasanov, S. Virtanen, A. Hakkala, and J. Isoaho, "Application of Large Language Models in Cybersecurity: A Systematic Literature Review," *IEEE Access*, vol. 12, pp. 176751–176778, 2024, doi: 10.1109/ACCESS.2024.3505983.
- [11] K. Abdulghaffar, N. Elmrabit, and M. Yousefi, "Enhancing Web Application Security through Automated Penetration Testing with Multiple Vulnerability Scanners," *Computers*, vol. 12, no. 11, Nov. 2023, doi: 10.3390/computers12110235.
- [12] X. Wu *et al.*, "CurriculumPT: LLM-Based Multi-Agent Autonomous Penetration Testing with Curriculum-Guided Task Scheduling," *Applied Sciences (Switzerland)*, vol. 15, no. 16, Aug. 2025, doi: 10.3390/app15169096.
- [13] A. Al-Sabbagh, E. Abdo, K. Samrouth, A. A. Alkhatib, and M. Elkhodr, "Autosecagent: a semi-automated AI-driven penetration testing framework through recursive memory and real-time RAG," *J. Supercomput.*, vol. 82, no. 5, p. 318, Apr. 2026, doi: 10.1007/s11227-026-08439-z.
- [14] R. Raman, P. Calyam, and K. Achuthan, "ChatGPT or Bard: Who is a better Certified Ethical Hacker?," *Comput. Secur.*, vol. 140, May 2024, doi: 10.1016/j.cose.2024.103804.
- [15] G. Deng *et al.*, "PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing," in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA: USENIX Association, Aug. 2024, pp. 847–864. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/deng>
- [16] A. Happe, A. Kaplan, and J. Cito, "LLMs as Hackers: Autonomous Linux Privilege Escalation Attacks," *Empir. Softw. Eng.*, vol. 31, no. 3, p. 70, May 2026, doi: 10.1007/s10664-025-10758-3.
- [17] E. Derner, K. Batistić, J. Zahálka, and R. Babuška, "A Security Risk Taxonomy for Prompt-Based Interaction with Large Language Models", doi: 10.1109/ACCESS.2024.0322000.
- [18] B. Kulambayev, G. Astabayeva, Z. Mukanova, K. Makhmetova, S. Mambetov, and S. Joldasbayev, "Design and Empirical Evaluation of a Four-Layer AI Agent Architecture for Automated Web Application Security Testing," *Engineering, Technology and Applied Science Research*, vol. 16, no. 2, pp. 33582–33588, Jan. 2026, doi: 10.48084/etasr.16879.
- [19] M. Mudassar Yamin, E. Hashmi, M. Ullah, and B. Katt, "Applications of LLMs for Generating Cyber Security Exercise Scenarios," *IEEE Access*, vol. 12, pp. 143806–143822, 2024, doi: 10.1109/ACCESS.2024.3468914.
- [20] A. C. Moreno *et al.*, "Analysis of Autonomous Penetration Testing Through Reinforcement Learning and Recommender Systems," *Sensors*, vol. 25, no. 1, p. 211, Jan. 2025, doi: 10.3390/s25010211.
- [21] H. Liu, C. Liu, X. Wu, Y. Qu, and H. Liu, "An Automated Penetration Testing Framework Based on Hierarchical Reinforcement Learning," *Electronics (Basel)*, vol. 13, no. 21, p. 4311, Nov. 2024, doi: 10.3390/electronics13214311.
- [22] A. Mechri, M. A. Ferrag, and M. Debbah, "SecureQwen: Leveraging LLMs for vulnerability detection in python codebases," *Comput. Secur.*, vol. 148, p. 104151, Jan. 2025, doi: 10.1016/j.cose.2024.104151.
- [23] A. Srivastava, "Deploy Long-Context Reasoning and Agentic Workflows with MiniMax M3 on NVIDIA Accelerated Infrastructure | NVIDIA Technical Blog." Accessed: Aug. 01, 2026. [Online]. Available: <https://developer.nvidia.com/blog/deploy-long-context-reasoning-and-agentic-workflows-with-minimax-m3-on-nvidia-accelerated-infrastructure/>
- [24] L. Wang *et al.*, "Automated Penetration Testing with LLM Agents and Classical Planning," Dec. 2025, [Online]. Available: <http://arxiv.org/abs/2512.11143>