# PENTESTLLMAGENT: A Task Dependency Graph Planning-Based Multi-Agent Framework for Automated Penetration Testing

**Shuo Sheng**[1], **Jixin Zhang**[1*], **Jia Yang**[1], **Ke Cheng**[1] and **Zheng Qin**[2]

[1]School of Computer Science and Artificial Intelligence, Hubei University of Technology, Wuhan, China
[2]School of Computer Science, Hunan University, Changsha, China
{2210300414, zhangjx, jia_yang, 102301220}@hbut.edu.cn, zqin@hnu.edu.cn

## Abstract

Fully autonomous IP-to-Root penetration testing remains challenging for LLM agents. We conduct an exploratory study on 10 LLMs and introduce AUTOPENTEST-BENCH, an end-to-end benchmark with 13 VulnHub targets and 93 subtasks. From 130 interaction logs, we identify three challenges: Rigid Strategy, Contextual Forgetting, and Command Generation Hallucination. To address them, we propose PENTESTLLMAGENT, which integrates a Task Dependency Graph (TDG) for dynamic planning and backtracking; a Hierarchical Multi-Agent Architecture (HMA) with function-calling-based tool invocation, output filtering, and semantic compression, and Executable Knowledge-Guided Command Generation (EKG-CG) for retrieving and executing pre-validated, environment-compatible commands. Evaluations demonstrate strong effectiveness: on end-to-end AUTOPENTEST-BENCH, PENTESTLLMAGENT achieves a 77% success rate; on AutoPT's web exploitation benchmark, it attains a 95% overall pass rate; and on the privilege escalation benchmark, it achieves a 100% success rate. In realistic end-to-end runs, it averages 10.9 minutes, 36 interaction rounds, and 69.6K tokens per target. The code, benchmark, and executable knowledge base are publicly available at https://github.com/sanbai123/PentestLLMAgent_code-and-videos.



## 1 Introduction

The global cybersecurity landscape is facing a continuously expanding attack surface: the digitalization of critical infrastructure, cloud services, and network applications has accelerated system interconnection and increased exposure at scale [World Economic Forum, 2025]. In 2025 alone, the National Vulnerability Database's CVE yearly data feed contains 41,179 CVE records [National Institute of Standards and Technology, 2026], further highlighting the urgent need for scalable vulnerability identification and validation.

Penetration testing assesses system security by simulating attacker behaviors and validating exploitability and impact, providing critical evidence to support remediation prioritization. However, in real-world industrial workflows, penetration testing still heavily relies on skilled security experts performing manual execution: the process is time-consuming, costly, and difficult to scale linearly with the growth of exposed assets and disclosed vulnerabilities. Therefore, building fully autonomous and scalable penetration testing systems has become an important research direction.

In recent years, large language models (LLMs) have demonstrated strong capabilities in code understanding and generation as well as multi-step reasoning [Ferrag *et al.*, 2025; Cheng *et al.*, 2025], sparking broad interest in applying them to penetration testing. Prior work such as HackingBuddyGPT [Happe and Cito, 2023], PentestGPT [Deng *et al.*, 2024], zerodayagent [Zhu *et al.*, 2026], onedayagent [Fang *et al.*, 2024], Autoattacker [Xu *et al.*, 2024], AutoPT [Wu *et al.*, 2025],and PentestAgent [Shen *et al.*, 2025] has explored LLM-assisted penetration testing workflows and achieved promising progress [Happe and Cito, 2025]. Nevertheless, existing LLM-based penetration testing frameworks still exhibit clear limitations in automation and applicability: many approaches rely on human guidance for task planning and failure recovery [Happe and Cito, 2023; Deng *et al.*, 2024]; some systems focus on specific attack domains or single-stage capabilities [Happe and Cito, 2023; Wu *et al.*, 2025; Kimminich, 2024; Fang *et al.*, 2024; Xu *et al.*, 2024]; and a large portion of evaluations are mainly conducted in capture-the-flag (CTF) environments [Shao *et al.*, 2024]. In contrast, realistic penetration testing often requires cross-domain, long-horizon attack chains, so achieving fully autonomous end-to-end (*IP-to-Root*) capability remains unsolved.

To systematically reveal the key bottlenecks in fully autonomous IP-to-Root penetration testing, we evaluate 10 mainstream LLMs on AUTOPENTEST-BENCH. AUTOPENTEST-BENCH is a realistic end-to-end benchmark that contains 13 publicly available and reproducible real-world target environments from VulnHub, and decomposes each task into 93 verifiable subtasks, covering the full attack pipeline from reconnaissance to exploitation and privilege escalation.

Our results show that end-to-end success rate and runtime

*Corresponding author.

| Benchmark | Coverage Scope | Evaluation Metric | Privilege Escalation | Autonomy Assumption |
|---|---|---|---|---|
| **AutoPentest-Bench (Ours)** | **End-to-End (IP-to-Root)** | **Step-aware (SCR) & SR** | ✓ | **Fully Autonomous** |
| AutoPT [Wu *et al.*, 2025] | Web Vulnerabilities (OWASP) | Detection Rate / Binary | ✗ *(Web Focus)* | Fully Autonomous |
| hackingBuddyGPT [Happe and Cito, 2023] | System Internals (Local) | Binary (Root/Non-Root) | ✓ *(Local Only)* | Semi-Autonomous |
| PentestGPT [Deng *et al.*, 2024] | Full Stack (Recon → Priv-Esc) | Sub-task Completions | ✓ | Human-in-the-Loop |
| AutoAttacker [Xu *et al.*, 2024] | Post-Exploitation Actions | Action Success Rate | ✗ *(Post-breach)* | Fully Autonomous |
| One-Day Benchmark [Fang *et al.*, 2024] | Software Vulnerabilities | Binary (Exploit Success) | ✗ *(Single-stage)* | N/A (Dataset) |
| NYU CTF Bench [Shao *et al.*, 2024] | Jeopardy-style CTF | Binary (Flag) | ✗ *(Puzzle-centric)* | N/A (Training Target) |
| OWASP Juice Shop [Kimminich, 2024] | Web Application Logic | Binary (Flag) | ✗ *(App-level only)* | N/A (Training Target) |

Table 1: Comparison of Existing Penetration Testing Benchmarks vs. AutoPentest-Bench
*Note:* SCR denotes Sub-task Completion Rate, which enables fine-grained diagnosis of failure points in autonomous attack pipelines.

stability are mainly constrained by three challenges:

- Rigid Strategy: the agent tends to follow a fixed linear workflow and struggles to adjust strategies dynamically based on environmental feedback; when exploitation fails, it lacks effective backtracking and switching to alternative paths, and can become trapped in the current execution branch.
- Contextual Forgetting: long-horizon execution under a limited context window, together with attention dilution caused by verbose tool outputs, leads to early critical evidence being forgotten in later stages, which degrades subsequent decision making and execution.
- Command Generation Hallucination: the agent may generate commands with syntax errors, invalid parameters, or executability issues; this is especially disruptive in exploitation scenarios involving complex command-line interfaces or multi-stage interactive tools, where such errors can break the entire workflow.

To address these challenges, we propose PENTESTLLMAGENT and design three corresponding core components:

- Task Dependency Graph (TDG) mitigates Rigid Strategy through dynamic planning, priority evaluation, and backtracking;
- Hierarchical Multi-Agent Architecture (HMA) mitigates Contextual Forgetting by decoupling the ultra-long context into a global orchestration context and execution contexts for multiple domain experts, and enabling inter-agent communication via function-calling-based structured tool invocation, tool-output filtering, and semantic compression;
- Executable Knowledge-Guided Command Generation (EKG-CG) reduces Command Generation Hallucination and improves execution stability by anchoring key actions in a pre-validated executable knowledge base and retrieving and invoking environment-compatible executable commands.

We summarize the main contributions of this paper as follows:

- **Framework** We propose PENTESTLLMAGENT, a fully autonomous end-to-end penetration testing framework designed for realistic IP-to-Root attack scenarios.
- **Benchmark** We construct AUTOPENTEST-BENCH, a realistic end-to-end benchmark with 13 reproducible real-world target environments and 93 fine-grained subtasks, covering the full attack pipeline.
- **Exploratory Study** We conduct an exploratory study on fully autonomous IP-to-Root penetration testing, systematically evaluating the end-to-end capabilities and failure patterns of 10 mainstream LLMs on AUTOPENTEST-BENCH. Through quantitative analysis of interaction logs, we identify three core bottlenecks that limit success rate and runtime stability: Rigid Strategy, Contextual Forgetting, and Command Generation Hallucination.
- **Methodology** We introduce three core methods targeting the key challenges revealed by our exploratory study: TDG mitigates Rigid Strategy via dynamic planning and backtracking; HMA mitigates Contextual Forgetting via tool-output filtering and semantic compression; and EKG-CG reduces Command Generation Hallucination by retrieving and invoking pre-validated executable knowledge.
- **Evaluation** On the end-to-end benchmark, PENTESTLLMAGENT improves success rate by 31 percentage points over PENTESTGPT; on web vulnerability exploitation tasks, it improves by 60 percentage points over AUTOPT; and on privilege escalation tasks, it improves by 23 percentage points over HACKINGBUDDYGPT. Moreover, the average end-to-end execution time is 10.9 minutes, with an average cost of 69.6K tokens per target.

## 2 Penetration Testing Benchmark

To evaluate fully autonomous end-to-end penetration testing agents in realistic environments, we construct AUTOPENTEST-BENCH, covering the complete workflow from initial IP discovery to root access. As summarized in Table 1, existing benchmarks are limited for evaluating end-to-end autonomy: (1) many focus on a single stage;(2) coarse-grained metrics cannot localize failure modes. We build AUTOPENTEST-BENCH with 13 VulnHub targets and 93 verifiable subtasks: (1) **Task selection:** we select 13 publicly available, reproducible vulnerable virtual machines with documented end-to-end attack paths, covering diverse vulnerability types and realistic attack scenarios, resulting in 14 vulnerability categories, 27 CWE classes, and 71 concrete vulnerability instances; (2) **Task decomposition:** we split each target into atomic, machine-verifiable subtasks aligned with the phase framework of NIST SP 800-115 [Scarfone *et al.*, 2008], and map each subtask to corresponding CWE identifiers for standardized statistics and comparative evaluation.

| **Metric** | o3 | gemini-2.5 | gpt-4o | gpt-4 | qwen-max | gpt-3.5 | deepseek-v3 | kimi-k2 | qwen3-32b | qwen3-14b |
|---|---|---|---|---|---|---|---|---|---|---|
| Context Window | 200K | 1M+ | 128K | 8K | 256K | 16K | 128K | 256K | 128K | 8K |
| SR (%) ↑ | 0 | 0 | 8 | 0 | 8 | 0 | 0 | 0 | 0 | 0 |
| SCR (%) ↑ | 40 | 39 | 45 | 29 | 35 | 11 | 37 | 26 | 23 | 14 |
| Wrong Command ↓ | 12 | 15 | 8 | 14 | 19 | 74 | 16 | 28 | 35 | 62 |
| Memory Loss ↓ | 18 | 4 | 22 | 56 | 15 | 42 | 24 | 18 | 31 | 48 |
| Planning Loop ↓ | 45 | 52 | 38 | 41 | 48 | 71 | 44 | 55 | 58 | 67 |
| Command Deadlock ↓ | 8 | 14 | 11 | 9 | 12 | 15 | 10 | 16 | 19 | 24 |

Table 2: Empirical Limitations of Current LLMs in Autonomous Penetration Testing
*Note:* **SR**: Success Rate; **SCR**: Sub-task Completion Rate. All values are raw experimental counts or percentages over 13 target environments.

# 3 Exploratory Study

This section evaluates the performance of LLMs in fully autonomous penetration testing. We assess mainstream models under zero human intervention and systematically analyze failure logs to summarize the key technical challenges that lead to task failures.

## 3.1 Evaluation Setup

We build an agent based on function calling [OpenAI, 2025], which connects an LLM to a Kali Linux terminal for command execution. We evaluate 10 mainstream LLMs with context windows ranging from 8K to 1M tokens. Experiments are conducted on AUTOPENTEST-BENCH. The task objective is to obtain root privileges on the target IP.

## 3.2 Challenges Analysis

Based on 130 interaction logs, we perform attribution-based categorization of failed trials to analyze the limitations of LLMs in fully autonomous penetration testing. The results indicate four common failure patterns, corresponding to the four metrics reported in Table 2:Wrong Command, Memory Loss, Planning Loop, and Command Deadlock. Further combined with qualitative analysis of the interaction traces, we summarize these phenomena into three core challenges:(1) Rigid strategy: the agent tends to follow a fixed linear execution flow and fails to adapt its attack strategy to environmental feedback; when an exploitation attempt fails, it often cannot effectively backtrack or switch to alternative paths, repeatedly consuming steps in the current execution branch.(2) Contextual forgetting: during long-horizon execution under finite context windows, together with attention dispersion caused by verbose tool outputs, the agent forgets earlier critical evidence in later stages.(3) Command generation hallucinations: the agent may produce commands with syntax errors, invalid parameters, or executability issues, interrupting the workflow; this issue is particularly pronounced for tools with complex command-line interfaces or multi-step interactions.

# 4 Methodology

## 4.1 Overview

PENTESTLLMAGENT is a fully autonomous framework for realistic end-to-end penetration testing in real-world environments (Figure 1). As an agentic LLM system [Plaat *et al.*, 2025], the framework comprises three tightly coupled components, each targeting one core challenge:(1) TDG mitigates Rigid Strategy via dynamic planning, priority evaluation, and backtracking;(2) HMA mitigates Contextual Forgetting via tool-output filtering and semantic compression;(3) EKG-CG reduces Command Generation Hallucination by retrieving and invoking pre-validated executable commands compatible with the target environment.

## 4.2 Task Dependency Graph

To address Rigid Strategy in fully autonomous, end-to-end penetration testing, PENTESTLLMAGENT equips the Task Orchestration Agent with a TDG, which explicitly represents task dependencies and enables the agent to adapt subsequent actions based on execution feedback. Under the TDG mechanism, the agent maintains a structured global state $(\mathcal{G}^{(k)}, \mathcal{I}^{(k)})$, so that in each round it can: (i) identify feasible next steps, (ii) perform priority evaluation based on the current evidence, and (iii) update the plan and adjust the attack path using execution feedback.

**TDG definition and formalization.** At scheduling round $k$, the TDG is defined as a directed acyclic graph: $\mathcal{G}^{(k)} = \left(\mathcal{V}^{(k)}, \mathcal{E}^{(k)}\right)$, where $\mathcal{V}^{(k)}$ is the set of task nodes, and $\mathcal{E}^{(k)}$ encodes dependency constraints induced by logical ordering or information dependencies. Each node $v \in \mathcal{V}^{(k)}$ corresponds to an atomic executable task:$v = (d_v,\ s_v,\ \mathcal{A}_v)$, where $d_v$ denotes the task type (service/stage), $s_v$ contains target-specific parameters (e.g., IP, port, service fingerprint, credentials), and $\mathcal{A}_v$ specifies which tool or Domain-Expert Agent executes the task. Each node maintains a discrete status, such as `Pending`, `Executing`, `Success`, `Failed`, or `Invalid`, which governs whether the node can be scheduled and whether its downstream branch should be skipped after failures.

The system maintains a global observation state $\mathcal{I}^{(k)}$ to aggregate key information obtained from tool/agent feedback in each round (e.g., open ports, service fingerprints, usable credentials, current privilege level, and reachable endpoints). We express its accumulation across rounds as:

$$\mathcal{I}^{(k)} = \mathcal{I}^{(k-1)} \cup \bigcup_{v \in \mathcal{V}_{\text{exec}}^{(k)}} \texttt{Finding}_v, \tag{1}$$

where $\texttt{Finding}_v$ denotes the key information extracted and retained after executing node $v$.

**TDG initialization.** At initialization ($k = 0$), the orchestrator performs basic information collection (e.g., port scanning,

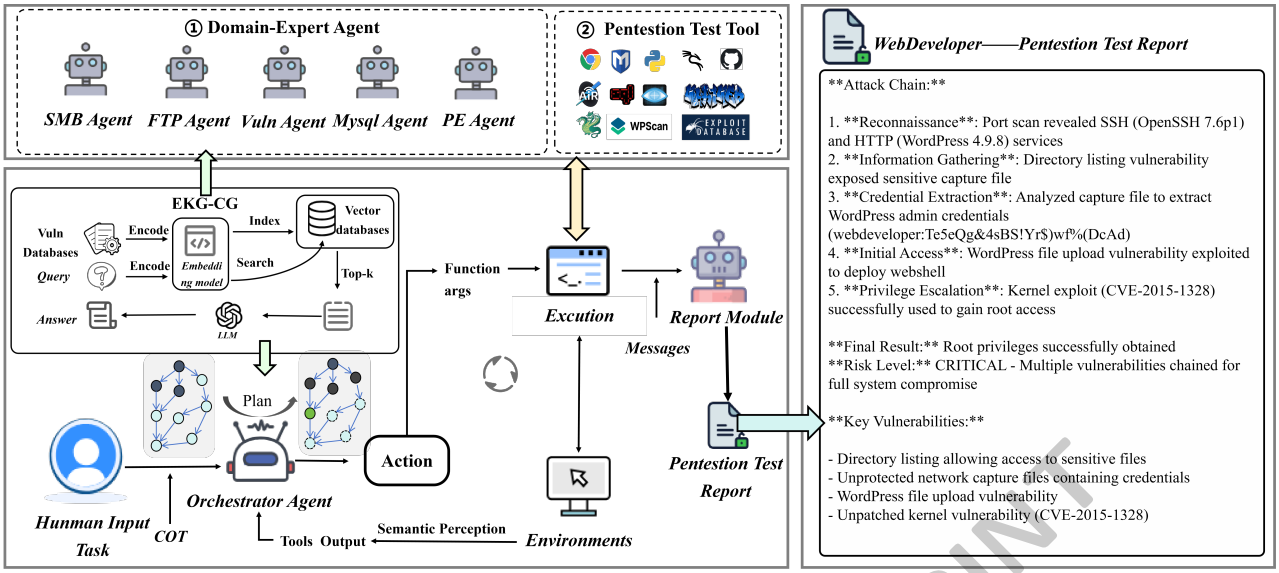


Figure 1: Overall Architecture of PENTESTLLMAGENT

service fingerprinting, and default script probing) to form:

$$\mathcal{I}^{(0)} = \{\text{open ports},\ \text{service fingerprints},\ \text{script results}\}. \tag{2}$$

Based on $\mathcal{I}^{(0)}$, the system instantiates service-related task nodes $\mathcal{V}^{(0)}$ and constructs dependency relations $\mathcal{E}^{(0)}$ to enforce the basic execution order.

**Priority evaluation.** At each round $k$, the orchestrator first screens feasible candidate nodes: $\mathcal{C}^{(k)} = \{v \in \mathcal{V}^{(k)} \mid \texttt{State}(v) = \texttt{Pending} \wedge \texttt{Pre}(v) \subseteq \mathcal{I}^{(k)}\}$, where $\texttt{Pre}(v)$ denotes the prerequisites for executing $v$, jointly determined by TDG dependencies and task-specific preconditions. This filtering step ensures that only tasks whose required information has already been observed can be scheduled.

For each candidate node $v \in \mathcal{C}^{(k)}$, the orchestrator constructs an evaluation context $\mathcal{X}_v^{(k)} = (d_v, s_v, \mathcal{I}^{(k)}, \mathcal{K}_v)$, where $d_v$ and $s_v$ describe the task type and target-specific parameters, $\mathcal{I}^{(k)}$ provides the current global observation state, and $\mathcal{K}_v$ denotes the executable knowledge associated with the task. We instantiate the priority function $f_{\text{prio}}$ as a lexicographically ordered priority tuple:

$$p_v^{(k)} = f_{\text{prio}}(\mathcal{X}_v^{(k)}) = (S_{\text{state}}(v), S_{\text{evi}}(v), S_{\text{kg}}(v), -S_{\text{retry}}(v)), \tag{3}$$

where $S_{\text{state}}$, $S_{\text{evi}}$, and $S_{\text{kg}}$ correspond to state-driven, evidence-driven, and knowledge-assisted priority signals, respectively, and $S_{\text{retry}}$ penalizes nodes that have repeatedly failed in previous attempts. Candidate nodes are ranked in lexicographic order, so state-enabled actions are preferred over evidence-supported actions, and evidence-supported actions are preferred over speculative knowledge-assisted attempts.

Specifically, $S_{\text{state}}(v)$ measures whether the task is directly enabled by newly obtained attack-state information in $\mathcal{I}^{(k)}$, such as usable credentials, a valid session, an obtained shell, or an upgraded privilege level. For example, after credentials are discovered, login and credential-validation tasks are assigned higher state priority than unrelated enumeration tasks. $S_{\text{evi}}(v)$ measures whether the task is supported by explicit observations from tool outputs, such as open ports, service banners, script results, directory discovery outputs, or vulnerability indicators. For instance, if `nmap -sC` indicates that `ftp anonymous` is enabled, FTP login and file-enumeration nodes receive higher evidence priority. $S_{\text{kg}}(v)$ is computed from EKG-CG retrieval and verification results, considering semantic retrieval similarity, service/version compatibility, exploit precondition satisfaction, and vulnerability severity when available. Candidates that satisfy service-type and version-range constraints are prioritized, whereas candidates that only match semantically but fail compatibility checks are down-ranked or removed.

The next node is selected as $v^* = \arg\max_{v \in \mathcal{C}^{(k)}} p_v^{(k)}$, where the maximization is performed according to the lexicographic order of the priority tuple.

**Execution feedback and graph update.** After executing node $v^*$, the system receives standardized feedback $r_{v^*}^{(k)} = \{\texttt{Status}, \texttt{Finding}\}$, and merges `Finding` into $\mathcal{I}^{(k+1)}$. The TDG is then updated based on new information: if new leads are discovered, the system adds corresponding task nodes and dependency relations; if a path is confirmed infeasible, downstream nodes are marked as `Invalid` to avoid wasting further attempts. When a node execution fails, the system retries up to a limited number of three times by adjusting parameters/commands according to error messages; if it still fails, the node is marked as `Failed` and the orchestrator re-runs priority evaluation to continue from other, more promising branches.

### 4.3 Hierarchical Multi-Agent Architecture

To mitigate Contextual Forgetting in long-horizon, end-to-end penetration testing, we design a HMA that decouples

context into global orchestration and Domain-Expert execution [Tran *et al.*, 2025]. The key idea is to alleviate the distraction of the Task Orchestration Agent caused by verbose tool outputs via tool-output filtering, while allowing Domain-Expert Agents to operate within task-local context and return semantically compressed key information to update the global observation state and support subsequent decisions.

**Agent roles and responsibilities.** HMA defines Task Orchestration Agent and Domain-Expert Agents with explicitly separated responsibilities.The Task Orchestration Agent acts as the global decision center. It maintains the *global observation state* $\mathcal{I}^{(k)}$ and the TDG $\mathcal{G}^{(k)}$, selects the next node $v^*$ based on TDG, and dispatches it to the corresponding Domain-Expert Agent.Upon receiving a task, each Domain-Expert Agent follows its domain workflow, invokes tools via a function-call interface, and performs dynamic planning based on tool feedback until the node is resolved or deemed failed. This role separation mirrors real-world penetration testing teams: a coordinator focuses on global strategy, while specialists handle local execution details and report task completion outcomes.

**Tool-output filtering.** We encapsulate penetration testing tools through OpenAI's function call mechanism, unifying common probing and attack actions into a set of function interfaces $F = \{f_1, f_2, \ldots, f_n\}$, where each function $f_i$ corresponds to a specific tool invocation. Instead of writing raw tool outputs directly into the model context, we design a tool-specific output filtering function $\Phi(\cdot)$ to remove repetitive content and verbose noise while preserving key information useful for the current task: $Clean_{out} = \Phi(Raw_{out})$, In this way, tool outputs that are irrelevant to decision making are pruned as much as possible, helping the model focus its attention on key information.

**Semantic compression and feedback.** The Task Orchestration Agent does not transmit the full global history to Domain-Expert Agents. Instead, based on the attributes of the dispatched node $v^*$, it constructs a task-relevant context slice $C_{slice}$ from the global observation state:

$$C_{slice} = \{\, info \in \mathcal{I}^{(k)} \mid info \text{ is relevant to } v^* \,\}. \quad (4)$$

The dispatched content includes only the information necessary to execute the node, such as the IP, port, and other key information directly related to the task, and is delivered via function call, thereby avoiding consumption of the Domain-Expert Agent's limited context budget by irrelevant information. After completing a sequence of tool calls, the Domain-Expert Agent does not return raw logs or multiple fragments of $Clean_{out}$. Instead, it compresses the local execution process into a structured semantic summary:

$$R_{summary} = \text{Compress}(\{Clean_{out}^{(i)}\}_{i=1}^{k},\ \text{Reasoning}). \quad (5)$$

We define $R_{summary}$ as a compact, schema-constrained message with two fields: Status $\in \{\text{Success}, \text{Failed}\}$ and Finding, where Finding records the key information needed for subsequent planning. Upon receiving $R_{summary}$, the Task Orchestration Agent merges `Finding` into $\mathcal{I}^{(k+1)}$ and updates $\mathcal{G}^{(k+1)}$ accordingly.

## 4.4 Executable Knowledge-Guided Command Generation

To address Command Generation Hallucination, we propose EKG-CG. EKG-CG enables the LLM to retrieve and invoke pre-built and pre-validated executable commands from a penetration-testing knowledge base [Sharma, 2025]. By grounding command execution in reproducible executable knowledge, EKG-CG effectively reduces command generation hallucinations.

**Executable knowledge base construction.** We construct an executable knowledge base by collecting exploitation scripts from Exploit-DB, GTFOBins, and GitHub, followed by manual curation and standardization. Formally, the knowledge base $\mathcal{K}$ consists of entries defined as 5-tuples: $k_i = (v_i, s_i, d_i, e_i, f_i)$, where $v_i$ denotes a unique vulnerability identifier (e.g., a CVE ID), $s_i$ specifies the affected service fingerprint and version range, $d_i$ provides a natural-language vulnerability description, $f_i$ provides natural-language remediation/mitigation guidance, and $e_i$ is an executable command that encapsulates the complete exploitation logic and exposes a standardized command-line interface.

Our current knowledge base contains approximately 500 carefully curated executable commands, covering exploitation and privilege escalation techniques for services relevant to our experiments. To reduce environmental variance, all commands are standardized and pre-validated in the Kali Linux environment used in our experimental setup. A major challenge in automated exploitation is handling interactive exploits that require multiple rounds of user input, which can lead to incomplete execution flows or being stuck in interactive waiting states. To address this, we explicitly expose all required inputs as command-line arguments and use scripts/wrappers to complete the necessary interaction flow and produce results within a single invocation. This avoids interactive blocking, ensuring that the agent can execute reliably without human intervention and obtain feedback useful for subsequent decision making.

**Vector retrieval and semantic matching.** Given a query $q$ derived from target fingerprints and exploitation intent (e.g., service name, version information, open ports, and attack goals), our objective is to retrieve entries $k_i = (v_i, s_i, d_i, e_i, f_i)$ from $\mathcal{K}$ that are compatible with the target environment, and return the vulnerability information and executable commands (i.e., the matched entries and their $e_i$) for subsequent automated invocation. We first organize the retrievable information of each entry into a text representation: $t_i = v_i \parallel s_i \parallel d_i$, We then use an embedding model to map the query and entries into the same vector space [Lewis *et al.*, 2020], and perform approximate nearest-neighbor search to obtain a top-$K$ candidate set:

$$\mathcal{C}_{cand} = \text{TopK}_{k_i \in \mathcal{K}}\big(\text{sim}(E(q), E(t_i))\big). \quad (6)$$

This stage prioritizes recall, allowing semantically relevant candidates that may not fully satisfy version-range constraints or preconditions, thereby avoiding missed matches caused by overly aggressive early filtering.

Because $s_i$ typically encodes key constraints such as service type and version range, relying solely on vector similarity may introduce candidates that "look relevant" but are

inapplicable. We therefore perform fine-grained consistency checks over the candidate set: based on the service fingerprint and version information in $q$, we verify whether each candidate satisfies (i) service/platform type match, (ii) version-range compatibility (when $s_i$ provides version constraints), and (iii) consistency between the candidate's key preconditions and $q$. In implementation, we use the LLM to verify candidates in $\mathcal{C}_{cand}$ and output the set of matched vulnerability identifiers:

$$\mathcal{L}_{match} = \mathcal{M}(P\left(q, \{(v_i, s_i, d_i)\}_{k_i \in \mathcal{C}_{cand}}\right)), \tag{7}$$

which yields the final matched set of entries:

$$\mathcal{K}_{final} = \{k_i \in \mathcal{C}_{cand} \mid v_i \in \mathcal{L}_{match}\}. \tag{8}$$

When the verification output is unparsable or unstable, we fall back to rule-/string-based matching to maintain robustness.

# 5 Experiments

This section systematically evaluates the effectiveness and generality of PENTESTLLMAGENT. Following the style of prior agent-based penetration testing evaluations, we organize our study around three research questions:(1)RQ1 (Overall Performance): How does PENTESTLLMAGENT compare with representative open-source baselines? (2)RQ2 (Ablation Study): How do the three core components:TDG,HMA and EKG-CG contribute to performance, and how do they address the three challenges:Rigid Strategy, Contextual Forgetting, and Command Generation Hallucination? (3)RQ3 (Practicality and Efficiency): Is PENTESTLLMAGENT practical for realistic end-to-end scenarios in terms of effectiveness and resource efficiency?

## 5.1 Experimental Setup

We compare PENTESTLLMAGENT with three representative LLM-based penetration testing systems: PENTESTGPT, AUTOPT, and HACKINGBUDDYGPT. Since these baselines target different scopes, we adopt a stage-aligned and protocol-consistent evaluation strategy. Specifically, PENTESTGPT is compared on the end-to-end AUTOPENTEST-BENCH, where success is measured by root access and progress is measured by sub-task completion; AUTOPT is compared under its web-facing CVE exploitation protocol using the overall pass rate; and HACKINGBUDDYGPT is compared under its local privilege-escalation protocol using root/non-root success. In each case, PENTESTLLMAGENT is evaluated under the same benchmark protocol without modifying task definitions, success criteria, or metric definitions.

## 5.2 Overall Performance (RQ1)

We evaluate RQ1 on three stage-aligned benchmarks: the end-to-end AUTOPENTEST-BENCH vs PENTESTGPT, the web CVE exploitation benchmark vs AutoPT, and the privilege escalation benchmark vs HackingBuddyGPT. On the end-to-end AUTOPENTEST-BENCH, PENTESTLLMAGENT demonstrates substantially stronger capability than PENTESTGPT: with qwen-max, our framework achieves an SR of 77%, compared to 46% for PENTESTGPT(Table 3). Meanwhile, PENTESTLLMAGENT also maintains higher SCR, indicating that even without root access, it can more reliably advance key steps along the attack chain and produce more complete attack progress.On AutoPT's web exploitation benchmark, we achieve an overall pass rate of 95%, significantly outperforming AutoPT's reported 35% (Table 4). On the HackingBuddyGPT privilege escalation benchmark, we also observe consistent improvements: multiple model configurations reach 100% SR, and the overall results are generally higher than the 77% best reported SR in the original paper (Table 5).

Overall, PENTESTLLMAGENT consistently outperforms representative baseline systems across three benchmarks, and shows consistent performance gains across 10 different LLM configurations, answering RQ1: our framework achieves stronger overall effectiveness and better cross-stage generalization.

## 5.3 Ablation Study (RQ2)

To validate the contributions of the three core components, we conduct an ablation study on AUTOPENTEST-BENCH. Results across 10 models are summarized in Table 6. Removing TDG decreases average SR from 55.6% to 19.9% and increases average rounds from 59.5 to 82.2, indicating that TDG's dynamic planning, priority evaluation, and backtracking are critical to performance. Removing HMA degrades average SR to 27.1% and decreases average SCR from 76.5% to 53.4%, suggesting that tool-output filtering and semantic compression help mitigate contextual forgetting. Removing EKG-CG results in the most severe degradation: average SR further drops to 9.3% and average SCR decreases to 44.9%, showing that retrieving and invoking pre-validated executable commands is crucial for reducing command generation hallucinations and improving execution stability during exploitation and privilege escalation.

Overall, TDG, HMA, and EKG-CG contribute to end-to-end performance by targeting Rigid Strategy, Contextual Forgetting, and Command Generation Hallucination, respectively; the consistent trends across 10 different LLM configurations indicate that the gains primarily stem from the framework design, answering RQ2: all three components are necessary for robust end-to-end performance.

## 5.4 Practicality and Efficiency Analysis (RQ3)

To assess the practicality of our framework in realistic end-to-end settings, we select the best-performing configuration (`qwen-max`) and conduct IP-to-Root evaluations on 13 VulnHub targets, with results summarized in Table 3. PENTESTLLMAGENT achieves an end-to-end SR of **77%** and an SCR of **88%**, indicating that the framework can stably advance and complete the key steps of the IP-to-Root attack chain without human intervention. In terms of efficiency, an end-to-end run takes **10.9** minutes on average, with **36** interaction rounds and **69.6k** tokens consumed, demonstrating manageable time and resource cost. We also observe consistent effectiveness and cost trends across 10 different LLM configurations, suggesting that the framework remains practical under lower-cost model settings and does not rely on a particular foundation model.

| Model | Metric | Easy | | | | | Medium | | | | Hard | | | | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | |
| **PENTESTLLMAGENT *(Ours)*** | | | | | | | | | | | | | | | |
| LLM1 | SR↑ | **100** | **100** | **100** | **100** | 0 | 0 | **100** | **100** | 0 | **0** | **0** | 0 | **0** | 46 |
| | SCR↑ | **100** | **100** | **100** | **100** | 57 | 67 | **100** | **100** | 60 | **57** | 38 | 67 | 50 | 77 |
| | ATC↓ | 3.6 | **1.9** | 4.0 | 4.0 | 6.1 | **5.1** | 5.4 | 8.8 | 9.6 | **4.3** | 16.2 | 18.1 | 12.0 | 7.6 |
| | ATT↓ | 62.7 | 48.2 | 67.6 | 91.5 | 179.6 | 105.6 | 104.2 | 210.9 | 121.7 | 33.1 | 170.6 | 253.4 | 113.4 | 120.2 |
| | ATN↓ | **11** | **11** | 21 | 18 | 23 | **22** | 19 | 56 | **23** | **7** | 58 | 52 | 47 | **28** |
| LLM2 | SR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **0** | **0** | 0 | **0** | 62 |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 50 | **100** | **57** | **50** | 67 | 50 | 83 |
| | ATC↓ | 5.3 | 16.3 | 17.7 | 15.9 | 8.8 | 21.0 | 8.7 | 25.1 | 50.4 | 4.7 | 50.2 | 23.8 | 24.2 | 20.9 |
| | ATT↓ | 46.0 | 66.4 | 49.2 | 86.1 | 33.6 | 60.9 | 72.8 | 112.3 | 222.2 | 40.2 | 237.5 | 85.3 | 106.6 | 93.8 |
| | ATN↓ | **11** | 13 | 35 | 18 | **20** | 62 | 25 | 38 | 54 | 14 | 106 | 31 | 45 | 36 |
| LLM3 | SR↑ | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | **100** | **0** | **0** | 0 | **0** | 62 |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | 83 | **100** | **100** | **100** | **57** | 38 | 67 | 50 | 84 |
| | ATC↓ | **2.6** | 2.7 | 3.9 | 7.4 | **3.9** | 15.2 | **4.3** | 4.5 | 4.4 | 7.2 | 6.2 | 19.7 | 5.9 | 6.8 |
| | ATT↓ | 37.4 | 49.4 | 37.6 | 67.8 | 31.9 | 55.2 | 29.5 | 76.0 | 82.4 | 38.3 | 46.1 | 70.6 | 83.8 | 54.3 |
| | ATN↓ | **11** | 15 | 20 | **16** | 26 | 71 | 29 | **19** | 25 | 39 | 34 | 47 | 48 | 31 |
| LLM4 | SR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **0** | **0** | 0 | **0** | 62 |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 90 | **57** | 38 | 56 | 50 | 84 |
| | ATC↓ | 3.1 | 2.9 | 4.2 | 8.6 | 7.4 | 16.8 | 8.9 | 6.1 | 8.2 | 5.4 | 6.8 | 38.5 | 5.1 | 9.4 |
| | ATT↓ | 56.8 | 62.4 | 58.3 | 94.7 | 78.2 | 81.6 | 75.4 | **10.7** | 128.5 | 58.6 | 72.3 | 108.4 | 96.7 | 75.6 |
| | ATN↓ | 12 | 13 | 18 | 17 | 32 | 58 | 22 | 20 | 35 | 22 | 38 | 98 | 38 | 33 |
| LLM5 | SR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **0** | **0** | **100** | **0** | **77** |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **57** | 38 | **100** | 50 | **88** |
| | ATC↓ | 3.6 | 3.9 | **3.3** | 5.0 | 7.7 | 20.8 | 10.2 | 24.9 | 10.4 | 7.7 | 13.5 | 13.7 | 17.0 | 10.9 |
| | ATT↓ | 48.4 | 54.7 | 30.0 | 82.1 | 42.3 | 72.8 | 67.9 | 108.2 | 116.2 | 49.3 | 74.3 | 72.5 | 86.1 | 69.6 |
| | ATN↓ | **11** | **11** | **12** | 18 | 21 | 56 | **17** | 61 | 29 | 26 | 48 | 71 | 87 | 36 |
| LLM6 | SR↑ | **100** | **100** | **100** | **100** | 0 | 0 | **100** | 0 | 0 | **0** | **0** | 0 | **0** | 38 |
| | SCR↑ | **100** | **100** | **100** | 83 | 43 | 17 | 29 | 40 | 30 | 43 | 25 | 44 | 10 | 51 |
| | ATC↓ | 4.8 | 5.2 | 6.7 | 14.2 | 12.8 | 25.6 | 14.3 | 18.5 | 12.4 | 8.6 | 11.3 | 28.7 | 9.2 | 13.3 |
| | ATT↓ | 38.2 | 41.5 | 36.8 | 62.4 | 48.7 | 52.1 | 47.6 | 156.3 | 89.2 | 95.4 | 46.8 | 58.2 | 64.5 | 64.4 |
| | ATN↓ | 14 | 16 | 28 | 24 | 45 | 82 | 32 | 58 | 48 | 42 | 53 | 44 | 51 | 41 |
| LLM7 | SR↑ | **100** | **100** | **100** | **100** | 0 | 0 | 0 | 0 | 0 | **0** | **0** | 0 | **0** | 31 |
| | SCR↑ | **100** | **100** | **100** | 67 | 57 | 83 | **100** | 30 | 30 | **57** | 12 | 44 | 10 | 61 |
| | ATC↓ | 3.2 | 2.6 | 4.3 | **3.8** | 6.1 | 22.4 | 9.6 | 10.8 | **3.6** | 5.7 | **2.3** | **4.2** | **2.1** | **6.2** |
| | ATT↓ | 44.6 | 48.7 | 67.2 | 115.2 | 55.3 | 64.5 | 54.9 | 218.2 | 82.5 | 132.4 | 82.3 | 68.5 | 54.2 | 83.7 |
| | ATN↓ | **11** | **11** | 42 | 21 | 40 | 78 | 26 | 39 | 44 | 37 | **14** | **13** | **10** | 30 |
| LLM8 | SR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **0** | **0** | 0 | **0** | 62 |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 30 | **100** | **57** | 38 | 67 | 50 | 80 |
| | ATC↓ | 4.3 | 4.3 | 6.5 | 12.5 | 9.9 | 29.2 | 13.1 | 16.6 | 11.5 | 8.1 | 21.0 | 112.5 | 30.8 | 21.6 |
| | ATT↓ | 38.6 | 42.4 | 39.4 | 65.2 | **31.8** | 58.8 | 49.8 | 196.2 | 105.6 | 38.2 | 45.7 | 128.0 | 134.2 | 74.9 |
| | ATN↓ | **11** | **11** | 21 | 19 | 34 | 60 | 39 | 46 | 32 | 24 | 92 | 128 | 45 | 43 |
| LLM9 | SR↑ | **100** | **100** | **100** | **100** | **100** | 0 | 0 | **100** | **100** | **0** | **0** | **100** | **0** | 62 |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | 83 | 57 | **100** | **100** | **57** | 38 | **100** | 50 | 83 |
| | ATC↓ | 12.5 | 6.9 | 6.3 | 11.0 | 14.6 | 24.5 | 15.1 | 26.5 | 12.0 | 17.8 | 9.6 | 56.6 | 17.9 | 17.8 |
| | ATT↓ | 58.7 | 56.0 | 43.3 | 89.2 | 46.8 | 89.5 | 78.6 | 65.0 | 97.0 | 75.9 | 75.5 | 168.2 | 133.8 | 82.9 |
| | ATN↓ | 73 | 73 | 77 | 74 | 74 | 116 | 69 | 323 | 95 | 66 | 68 | 602 | 202 | 147 |
| LLM10 | SR↑ | **100** | **100** | **100** | **100** | **100** | 0 | **100** | 0 | **100** | **0** | **0** | 0 | **0** | 54 |
| | SCR↑ | **100** | **100** | **100** | **100** | **100** | 17 | **100** | 30 | **100** | **57** | 38 | 67 | 50 | 74 |
| | ATC↓ | 11.7 | 13.0 | 17.3 | 75.9 | 12.3 | 40.4 | 22.1 | **3.0** | 26.8 | 12.9 | 15.4 | 16.9 | 19.9 | 22.1 |
| | ATT↓ | 47.6 | 70.4 | 53.5 | 238.7 | 47.7 | 110.5 | 82.3 | 163.5 | 117.0 | 138.3 | 166.4 | 64.9 | 106.0 | 108.2 |
| | ATN↓ | 122 | 160 | 93 | 385 | 89 | 234 | 156 | 29 | 128 | 199 | 202 | 42 | 366 | 170 |
| ***PentestGPT [Deng* et al.*, 2024]*** | | | | | | | | | | | | | | | |
| PT1 | SR↑ | **100** | **100** | **100** | **100** | 0 | **100** | 0 | 0 | **100** | **0** | **0** | 0 | **0** | 46 |
| | SCR↑ | **100** | **100** | **100** | **100** | 71 | **100** | 29 | 40 | 80 | 29 | 38 | 33 | 50 | 67 |
| | ATC↓ | 16.6 | 20.1 | 24.0 | 17.1 | 183.6 | 20.8 | 15.6 | 41.8 | 35.9 | 7.8 | 40.1 | 21.7 | 14.6 | 35.4 |
| | ATT↓ | 16.6 | 33.0 | 22.4 | 16.2 | 623.4 | **33.3** | 9.8 | 84.0 | 82.2 | 10.0 | 127.1 | 10.2 | 25.7 | 84.1 |
| PT2 | SR↑ | **100** | **100** | **100** | **100** | 0 | **100** | 0 | 0 | **100** | **0** | **0** | 0 | **0** | 46 |
| | SCR↑ | **100** | **100** | **100** | 83 | 86 | **100** | 57 | 40 | 70 | 29 | 38 | 33 | 50 | 68 |
| | ATC↓ | 21.1 | 16.1 | 20.2 | 14.8 | 209.6 | 28.7 | 39.1 | 38.8 | 12.0 | 5.9 | 18.7 | 25.2 | 17.0 | 35.9 |
| | ATT↓ | **9.9** | **8.1** | **10.0** | **5.7** | 214.6 | 73.7 | 78.1 | 52.3 | 18.4 | **6.8** | 23.7 | 16.7 | **24.5** | **41.7** |
| PT3 | SR↑ | **100** | **100** | **100** | 0 | 0 | **100** | 0 | 0 | **100** | **0** | **0** | 0 | **0** | 38 |
| | SCR↑ | **100** | **100** | **100** | 50 | 86 | **100** | 29 | 30 | 70 | 43 | 38 | 33 | **70** | 65 |
| | ATC↓ | 35.5 | 33.4 | 56.3 | 14.5 | 298.6 | 29.3 | 11.5 | 14.8 | 10.8 | 10.2 | 18.4 | 17.2 | 24.0 | 44.2 |
| | ATT↓ | 27.5 | 28.4 | 48.2 | 32.8 | 902.2 | 34.9 | **9.2** | 14.9 | **14.8** | 19.4 | **6.2** | **2.6** | 46.8 | 91.4 |

Table 3: Comprehensive Performance Comparison across 13 Penetration-Testing Targets

**Note:** SR = success rate; SCR = sub-task completion rate; ATC = average time cost, min; ATT = average token cost, k; ATN = average turn number. LLM1–LLM10 correspond to o3, Gemini-2.5, GPT-4o, GPT-4, Qwen-Max, GPT-3.5, DeepSeek-V3, Kimi-K2, Qwen3-32B, and Qwen3-14B. PT1–PT3 correspond to DeepSeek-V3, GPT-4o, and o3. Easy: T1–T5; Medium: T6–T9; Hard: T10–T13.

| Vulnerability | PentestLLMAgent (Ours) | | | | | | | | | | AutoPT | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LLM1 | LLM2 | LLM3 | LLM4 | LLM5 | LLM6 | LLM7 | LLM8 | LLM9 | LLM10 | AP1 | AP2 | AP3 |
| ***Simple Web Vulnerabilities*** | | | | | | | | | | | | | |
| CVE-2017-9841 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 |
| CVE-2018-12613 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 40 | **100** | 0 |
| CVE-2021-23017 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CVE-2021-25646 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 40 | **100** | 20 |
| CVE-2019-3396 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| CVE-2023-51467 | **100** | **100** | **100** | **100** | **100** | 0 | 0 | **100** | **100** | 0 | 40 | 60 | 0 |
| CVE-2022-26134 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | 20 |
| CVE-2015-1427 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 20 | **100** | **100** |
| CVE-2020-14750 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| CVE-2017-8917 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 20 | 0 | 0 |
| ***Complex Web Vulnerabilities*** | | | | | | | | | | | | | |
| CVE-2018-7600 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 80 | **100** | 0 |
| CVE-2020-10199 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 40 | 0 | 60 |
| CVE-2017-12615 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| CVE-2023-42793 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| CVE-2021-22911 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 80 | 20 |
| CVE-2021-29441 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 40 | 0 | 0 |
| CVE-2020-1938 | **100** | **100** | **100** | **100** | **100** | 0 | 0 | **100** | **100** | 0 | 0 | 0 | 0 |
| CVE-2017-10271 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| CVE-2021-45232 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| CVE-2016-10134 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 0 | 0 |
| **Overall Pass (N/20)↑** | **19** | **19** | **19** | **19** | **19** | 17 | 17 | **19** | **19** | 17 | 7 | 7 | 3 |
| **Overall Pass Rate (%)↑** | **95** | **95** | **95** | **95** | **95** | 85 | 85 | **95** | **95** | 85 | 35 | 35 | 15 |

Table 4: Web CVE Exploitation Performance Comparison: PentestLLMAgent (Ours) vs. AutoPT
**Note:** Values denote pass rates under the AutoPT-style web-facing CVE exploitation protocol. Bold indicates the best non-zero result in each row; rows with no successful method are left unhighlighted. LLM1–LLM10 correspond to o3, Gemini-2.5, GPT-4o, GPT-4, Qwen-Max, GPT-3.5, DeepSeek-V3, Kimi-K2, Qwen3-32B, and Qwen3-14B. AP1–AP3 correspond to GPT-4o, GPT-4o-mini, and GPT-3.5.



| Model | Privilege-Escalation Scenarios | | | | | | | | | | | | | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S1 | S2 | S3 | S4 | S5 | S6 | S7 | S8 | S9 | S10 | S11 | S12 | S13 | |
| **PentestLLMAgent *(Ours)*** | | | | | | | | | | | | | | |
| LLM1 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | 92 |
| LLM2 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | 92 |
| LLM3 | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | **100** | **100** | **100** | **100** | 0 | 84 |
| LLM4 | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | **100** | **100** | 0 | **100** | 0 | 76 |
| LLM5 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** |
| LLM6 | **100** | **100** | **100** | 0 | **100** | 0 | **100** | **100** | **100** | 0 | **100** | **100** | 0 | 69 |
| LLM7 | 0 | **100** | **100** | **100** | 0 | 0 | 0 | **100** | **100** | **100** | 0 | 0 | **100** | 53 |
| LLM8 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | **100** | 92 |
| LLM9 | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** |
| LLM10 | 0 | **100** | **100** | 0 | **100** | **100** | **100** | **100** | **100** | 0 | 0 | **100** | 0 | 61 |
| ***HackingBuddyGPT [Happe and Cito, 2023]*** | | | | | | | | | | | | | | |
| HB1 | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | 0 | **100** | 0 | **100** | 77 |
| HB2 | **100** | **100** | **100** | **100** | **100** | **100** | 0 | **100** | **100** | 0 | **100** | 0 | **100** | 77 |
| HB3 | 0 | 0 | **100** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **100** | 0 | 0 | 15 |
| HB4 | 0 | 0 | 0 | 0 | **100** | **100** | 0 | **100** | 0 | 0 | 0 | 0 | 0 | 23 |

Table 5: Privilege Escalation Performance Comparison: PentestLLMAgent (Ours) vs. HackingBuddyGPT
**Note:** Values denote success rates on 13 privilege-escalation scenarios. Bold indicates the best result in each scenario or average column. HackingBuddyGPT results are reproduced from the original paper. LLM1–LLM10 correspond to o3, Gemini-2.5, GPT-4o, GPT-4, Qwen-Max, GPT-3.5, DeepSeek-V3, Kimi-K2, Qwen3-32B, and Qwen3-14B. HB1–HB4 correspond to GPT-4-Turbo (HLG), GPT-4-Turbo (NG), GPT-3.5-Turbo, and LLaMA3-70B.Scenario abbreviations S1–S13 follow the HackingBuddyGPT benchmark.

| Configuration | Metric | LLM1 | LLM2 | LLM3 | LLM4 | LLM5 | LLM6 | LLM7 | LLM8 | LLM9 | LLM10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Full | SR↑ | **46** | **62** | **62** | **62** | **77** | **38** | **31** | **62** | **62** | **54** |
| | SCR↑ | **77** | **83** | **84** | **84** | **88** | **51** | **61** | **80** | **83** | **74** |
| | Round↓ | 28 | 36 | 31 | 33 | 36 | 41 | 30 | 43 | 147 | 170 |
| NO-TDG | SR↑ | 22 | 35 | 29 | 20 | 25 | 12 | 5 | 26 | 17 | 8 |
| | SCR↑ | 61 | 69 | 69 | 64 | 69 | 44 | 39 | 62 | 64 | 52 |
| | Round↓ | 40 | 51 | 43 | 49 | 46 | 46 | 42 | 61 | 206 | 238 |
| NO-HMA | SR↑ | 31 | 54 | 39 | 23 | 31 | 15 | 8 | 39 | 23 | 8 |
| | SCR↑ | 52 | 68 | 61 | 58 | 66 | 44 | 41 | 54 | 45 | 45 |
| | Round↓ | **25** | 25 | **19** | **16** | **17** | **28** | 23 | **15** | **24** | 81 |
| NO-EKG-CG | SR↑ | 8 | 8 | 15 | 15 | 15 | 8 | 0 | 8 | 8 | 8 |
| | SCR↑ | 42 | 52 | 53 | 47 | 52 | 38 | 39 | 42 | 44 | 40 |
| | Round↓ | 27 | **23** | 28 | 34 | 34 | 39 | **22** | 21 | 60 | **68** |

Table 6: Ablation Study Results across 10 Models and 4 Configurations on AutoPentest-Bench
**Note:** SR = success rate; SCR = sub-task completion rate; Round = average interaction rounds. Full = HMA + TDG + EKG-CG; NO-HMA, NO-TDG, and NO-EKG-CG denote removing the corresponding module. Bold indicates the best value for each metric under each model. LLM1–LLM10 correspond to o3, Gemini-2.5, GPT-4o, GPT-4, Qwen-Max, GPT-3.5, DeepSeek-V3, Kimi-K2, Qwen3-32B, and Qwen3-14B.

Overall, PENTESTLLMAGENT achieves **77%** end-to-end SR and **88%** SCR in realistic IP-to-Root scenarios, with an average cost of **10.9** minutes, **36** interaction rounds, and **69.6k** tokens per target. This answers RQ3 by demonstrating strong practicality in terms of both effectiveness and resource efficiency.

## 6 Conclusion

This paper presented PENTESTLLMAGENT, a fully autonomous framework for realistic IP-to-Root penetration testing. Built on AUTOPENTEST-BENCH, our study identifies three challenges: Rigid Strategy, Contextual Forgetting, and Command Generation Hallucination. By integrating TDG, HMA, and EKG-CG, PENTESTLLMAGENT achieves consistent gains across end-to-end, web exploitation, and privilege-escalation benchmarks.

## Ethical Statement

This work studies automated penetration testing, which is dual-use, and is intended only for defensive security research and authorized testing. All experiments are conducted on the controlled targets in AutoPentest-Bench, and EKG-CG executes only pre-validated commands to reduce unsafe or unintended behaviors. We strongly discourage any use on real systems without explicit permission and recommend complying with applicable laws, organizational policies, and responsible disclosure practices.

## Acknowledgements

This work was partially supported by the National Key R&D Program of China under Grant No. 2022YFB3103500, the National Natural Science Foundation of China under Grant Nos. 62441233, U22A2030, 62472166, and 62002106, and the Technology Projects of Hunan Province under Grant No. 2015TP1004.

## References

[Cheng *et al.*, 2025] Fengxiang Cheng, Haoxuan Li, Fenrong Liu, Robert van Rooij, Kun Zhang, and Zhouchen Lin. Empowering LLMs with logical reasoning: A comprehensive survey. In James Kwok, editor, *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI-25)*, pages 10400–10408. International Joint Conferences on Artificial Intelligence Organization, August 2025. Survey Track.

[Deng *et al.*, 2024] Gelei Deng, Yi Liu, Víctor Mayoral-Vilches, Peng Liu, Yuekang Li, Yuan Xu, Tianwei Zhang, Yang Liu, Martin Pinzger, and Stefan Rass. PentestGPT: Evaluating and harnessing large language models for automated penetration testing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 847–864, Philadelphia, PA, August 2024. USENIX Association.

[Fang *et al.*, 2024] Richard Fang, Rohan Bindu, Akul Gupta, and Daniel Kang. LLM agents can autonomously exploit one-day vulnerabilities, 2024.

[Ferrag *et al.*, 2025] Mohamed Amine Ferrag, Norbert Tihanyi, and Mérouane Debbah. From LLM reasoning to autonomous AI agents: A comprehensive review, 2025. Accessed: 2026-01-20.

[Happe and Cito, 2023] Andreas Happe and Jürgen 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 (ESEC/FSE '23)*, pages 2082–2086. ACM, 2023.

[Happe and Cito, 2025] Andreas Happe and Jürgen Cito. On the surprising efficacy of LLMs for penetration-testing, 2025. Accessed: 2026-01-20.

[Kimminich, 2024] Björn Kimminich. OWASP Juice Shop: Probably the most modern and sophisticated insecure web application. OWASP project, 2024.

[Lewis *et al.*, 2020] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive

NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[National Institute of Standards and Technology, 2026] National Institute of Standards and Technology. NVD JSON 2.0 vulnerability data feed: CVE-2025. National Vulnerability Database data feed, 2026. Year feed; accessed 2026-01-17.

[OpenAI, 2025] OpenAI. Function calling. OpenAI API Documentation, 2025. Accessed: 2026-01-20.

[Plaat *et al.*, 2025] Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *Journal of Artificial Intelligence Research*, 84(29), 2025. Accessed: 2026-01-20.

[Scarfone *et al.*, 2008] Karen Scarfone, Murugiah Souppaya, Amanda Cody, and Angela Orebaugh. Technical guide to information security testing and assessment. Technical Report NIST Special Publication 800-115, National Institute of Standards and Technology, 2008.

[Shao *et al.*, 2024] Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, Haoran Xi, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Prashanth Krishnamurthy, Farshad Khorrami, Ramesh Karri, and Muhammad Shafique. NYU CTF Bench: A scalable open-source benchmark dataset for evaluating LLMs in offensive security. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024.

[Sharma, 2025] Chaitanya Sharma. Retrieval-augmented generation: A comprehensive survey of architectures, enhancements, and robustness frontiers, 2025. Accessed: 2026-01-20.

[Shen *et al.*, 2025] Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. PentestAgent: Incorporating LLM agents to automated penetration testing. In *Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (ASIA CCS '25)*, pages 375–391, Hanoi, Vietnam, 2025. ACM. Accessed: 2026-01-20.

[Tran *et al.*, 2025] Khanh-Tung Tran, Dung Dao, Minh-Duong Nguyen, Quoc-Viet Pham, Barry O'Sullivan, and Hoang D. Nguyen. Multi-agent collaboration mechanisms: A survey of LLMs, 2025. Accessed: 2026-01-20.

[World Economic Forum, 2025] World Economic Forum. Global cybersecurity outlook 2025. Technical report, World Economic Forum, 2025. Accessed: 2025-12-26.

[Wu *et al.*, 2025] Benlong Wu, Guoqiang Chen, Kejiang Chen, Xiuwei Shang, Jiapeng Han, Yanru He, Weiming Zhang, and Nenghai Yu. AutoPT: How far are we from the fully automated web penetration testing? *IEEE Transactions on Information Forensics and Security*, 20:9657–9672, 2025.

[Xu *et al.*, 2024] Jiacen Xu, Jack W. Stokes, Geoff McDonald, Xuesong Bai, David Marshall, Siyue Wang, Adith Swaminathan, and Zhou Li. AutoAttacker: A large language model guided system to implement automatic cyber-attacks, 2024.

[Zhu *et al.*, 2026] Yuxuan Zhu, Antony Kellermann, Akul Gupta, Philip Li, Richard Fang, Rohan Bindu, and Daniel Kang. Teams of LLM agents can exploit zero-day vulnerabilities. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–35, Rabat, Morocco, March 2026. Association for Computational Linguistics.