xOffense: An AI-driven autonomous penetration testing framework with offensive knowledge-enhanced LLMs and multi agent systems

Phung Duc Luong ^{©a,b}, Le Tran Gia Bao ^{©a,b}, Nguyen Vu Khai Tam ^{©a,b}, Dong Huu Nguyen Khoa ^{©a,b}, Nguyen Huu Quyen ^{©a,b}, Van-Hau Pham ^{©a,b}, Phan The Duy ^{©a,b,*}

> ^aInformation Security Lab, University of Information Technology, Ho Chi Minh City, Vietnam ^bVietnam National University Ho Chi Minh City, Ho Chi Minh City, Vietnam

Abstract

Penetration testing (pentest) is essential for assessing the security of computer systems, yet conventional automated approaches using Machine Learning (ML), Deep Learning (DL) or Reinforcement Learning (RL) remain limited by simplified action spaces, high computational costs and weak reasoning across the multi-stage process of reconnaissance, vulnerability analysis and exploitation. Recent frameworks such as PentestGPT, VulnBot attempt to leverage large language models (LLMs) but face challenges of high cost, scalability and adaptability in complex workflows. This motivates the transition from monolithic models toward agentic Al systems, where multiple specialized agents collaborate to complete pentest tasks with higher efficiency and accuracy. This work introduces xOffense, an Al-driven, multi-agent penetration testing framework that shifts the process from labor-intensive, expert-driven manual efforts to fully automated, machine-executable workflows capable of scaling seamlessly with computational infrastructure. At its core, xOffense leverages a fine-tuned, mid-scale open-source LLM (Qwen3-32B) to drive reasoning and decision-making in penetration testing. The framework assigns specialized agents to reconnaissance, vulnerability scanning, and exploitation, with an orchestration layer ensuring seamless coordination across phases. Fine-tuning on Chain-of-Thought penetration testing data further enables the model to generate precise tool commands and perform consistent multi-step reasoning. We evaluate xOffense on two rigorous benchmarks: AutoPenBench and Al-Pentest-Benchmark. The results demonstrate that xOffense consistently outperforms contemporary methods, achieving a sub-task completion rate of 79.17%, decisively surpassing leading systems such as VulnBot and PentestGPT. These findings highlight the potential of domina-dapted mid-scale LLMs, when embedded within structured multi-agent orchestration, to deliver superior, cost-efficient, and reproducible solutions for autonomous penetration testing.

Introd

As networks grow in scale and complexity, the gap between the appearance of new vulnerabilities and the ability of security professionals to detect and remediate them is widening. This growing imbalance underscores the urgent necessity for automated and intelligent penetration testing solutions.

Email addresses: 21522312@gm.uit.edu.vn (Phung Duc Luong @), 22520105@gm.uit.edu.vn (Le Tran Gia Bao), 22521293@gm.uit.edu.vn (Nguyen Vu Khai Tam 0), 23520734@gm.uit.edu.vn (Dong Huu Nguyen Khoa 6), quyennh@uit.edu.vn (Nguyen Huu Quyen [6]), haupv@uit.edu.vn (Van-Hau Pham), duyptQuit.edu.vn (Phan The Duy)

model penetration testing as a sequential decision-making problem in partially observable environments, enabling agents to autonomously explore and learn effective attack strategies through interaction and reward-based learning. RL agents can, in principle, discover novel attack paths, yet in practice they face two key obstacles: (i) their action space must be heavily simplified (e.g., "scan port", "exploit CVE-xxx"), and (ii) training requires a large number of environment interactions that are expensive to obtain and seldom transfer between real networks. Consequently, even state-of-the-art RL-based pentesters achieve modest coverage and require significant engineering to integrate new tools or protocols.

These limitations illustrate that purely Deep Learning (DL)

September 17, 2025 Preprint submitted to Elsevier

^{*}Corresponding author

or RL-based automation is insufficient for the inherently multiphase and dynamic nature of penetration testing. To overcome this, recent works have turned toward AI agent-based paradigms, in which multiple specialized agents collaborate to emulate the workflow of human red teams. In such systems, each agent assumes a distinct role: a *Reconnaissance Agent* focuses on host and service discovery, a *Vulnerability Analysis Agent* correlates findings with CVE and CWE knowledge bases, and an *Exploitation Agent* generates and tests candidate payloads. This agent-oriented decomposition enables modularity, context retention across phases, and the possibility of scaling to complex attack paths that traditional ML/DL/RL pipelines cannot handle.

Recent advancements in Large Language Models (LLMs) have opened new possibilities for automating penetration testing. Leveraging their strong reasoning and code generation capabilities, LLMs have been adopted in several research prototypes such as PentestGPT [7], PentestAgent [8], and VulnBot [9], where models assist or autonomously conduct reconnaissance, scanning, and exploitation. In particular, VulnBot represents a major step forward: it frames penetration testing as a collaborative workflow between specialized LLM agents guided by a Penetration Task Graph (PTG), enabling the simulation of expert-level pentesting with limited or no human intervention. Empirical results from benchmarks like AutoPenBench [10] and AI-Pentest-Benchmark [11] have validated VulnBot's capacity to outperform other automated methods in structured testing environments.

However, most of these systems rely heavily on extremely large or commercial LLMs, such as GPT-40, LLaMA3-70B, or DeepSeek-V3. Despite their capabilities, these models present significant operational hurdles—including high resource consumption, costly API dependencies, and limited adaptability to domain-specific fine-tuning. Additionally, their general-purpose nature often leads to hallucinations, loss of context across phases, or poor command translation in complex penetration workflows. As such, there is a pressing need to explore whether smaller, fine-tuned open-source models can serve as more efficient, specialized alternatives—offering better controllability, lower cost, and targeted reasoning.

This work is motivated by two core observations. First, while large LLMs have demonstrated strong potential in security domains, scale alone does not guarantee effectiveness, particularly when models are deployed in structured, multi-step tasks like penetration testing. Despite their size, large-scale LLMs still suffer from context loss across phases, generate incorrect tool usage, and require significant human supervision. Second, most current systems adopt LLMs as black-box assistants or instruction followers, without integrating deeper taskspecific guidance or domain adaptation. To address these limitations, we explore an alternative paradigm: leveraging a midsized, domain-adapted LLM that is explicitly trained for penetration testing tasks. We present xOffense, a refined evolution of the VulnBot framework that substitutes its dependence on large, general-purpose models with a fine-tuned Qwen3-32B [12], a 32-billion-parameter open-source language model. Through dedicated training on penetration testing workflows, including vulnerability scanning, exploit crafting, and security tool interaction, xOffense achieves sharper task alignment, enhanced operational fidelity, and greater adaptability in nuanced or low-visibility environments.

Beyond simply replacing the core language model, xOffense also incorporates a context-aware prompting scheme we refer to as grey-box prompting. In this setup, agents are equipped with partial system insights, such as protocol hints, observed services, or prior scan summaries, enabling them to make more informed decisions without relying on full system disclosure. This strategy preserves the operational constraints of black-box testing while offering minimal structured guidance, striking a balance between realism and agent effectiveness. By preserving VulnBot's three-phase pipeline—reconnaissance, scanning, and exploitation—xOffense ensures compatibility with existing workflows, facilitates direct benchmarking, and provides a robust foundation for comparative evaluation.

In this paper, we present the design, implementation, and evaluation of xOffense, a lightweight, domain-adaptive, and highly effective autonomous penetration testing system. Our key contributions are as follows:

- An AI-driven multi-agent penetration testing system.
 We propose a novel agent-based framework in which specialized agents collaborate to cover all critical phases of penetration testing—reconnaissance, vulnerability analysis, and exploitation. This design emulates the workflow of human red teams, ensures modularity across tasks, and enables coherent orchestration of complex attack paths in an autonomous manner.
- A domain-adapted mid-scale LLM. At the core of our system lies Qwen3-32B, a 32B-parameter open-source model fine-tuned with Chain-of-Thought penetration testing data. This adaptation empowers the model with precise multi-phase reasoning, accurate tool command generation, and strong adaptability in complex exploitation workflows.
- Grey-box phase prompting. We introduce a contextaware prompting mechanism that selectively integrates environmental cues—such as observed protocols, discovered services, and prior scan outputs—into the agent reasoning process. This strategy strikes a balance between black-box and white-box testing, reducing context loss and improving continuity across phases.
- Extensive empirical validation. We conduct rigorous evaluations of xOffense on AutoPenBench and AI-Pentest-Benchmark, demonstrating state-of-the-art performance in both synthetic and real-world penetration testing scenarios. The system achieves superior task and sub-task completion rates compared to prior methods, confirming the effectiveness of multi-agent orchestration and domain-adapted LLMs.

The remainder of this paper is organized as follows. Section 2 reviews prior studies on penetration testing and automation approaches, while Section 3 introduces the fundamental

concepts that underpin automated systems. Section 4 describes the architecture of the proposed xOffense framework, including its fine-tuned Qwen3-32B model, grey-box prompting strategy, and multi-agent orchestration. We describe the experimental settings and benchmark datasets, evaluation metrics in Section 5. In Section 6 empirical results on two benchmarks and real-world exploitation scenarios are reported and analyzed. Section 7 discusses potential threats to validity and their implications for generalizability. Finally, Section 8 concludes the paper and outlines directions for future research.

2. Related work

2.1. Pre-LLM automation and RL

Deterministic orchestrators, such as DeepExploit, integrate scanners and exploit frameworks (e.g., Metasploit, Nmap, Nikto, WPScan) but rely on rigid rules and shallow evidence fusion, limiting adaptability to dynamic attack scenarios [3, 4, 13, 14, 15]. RL-based agents formulate pentesting as a Partially Observable Markov Decision Process (POMDP) with reward shaping, providing a principled approach to automating attack strategies [5, 6, 16]. Notably, the Raiju framework has made significant strides in automating post-exploitation tasks by leveraging RL algorithms, specifically Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) [17]. Integrated with Metasploit, Raiju trains specialized agents to perform tasks such as privilege escalation, hashdump gathering, and lateral movement in real-world environments, achieving a success rate exceeding 84% across diverse attack types in four tested environments. However, RL-based approaches, including Raiju, face two primary challenges: (i) the need for extensive state and action space engineering, and (ii) limited cross-target transferability without costly retraining. These limitations highlight the need for more adaptive methods, such as LLMs, to enhance efficiency and generalization in penetration testing.

2.2. LLM single-/few-agent pipelines

PentestGPT demonstrates a modular, self-interacting scaffold where an LLM plans, parses tool outputs, and synthesizes commands; this closes the perception ←action loop while mitigating context loss via summarization [7]. AutoAttacker focuses on *post-breach* realism with shell/Metasploit control across Windows/Linux, executing multi-step attacks [18]. Both highlight that language-grounded synthesis and disciplined tool use can automate substantial portions of a cyber kill-chain, yet often rely on large backbones and ad-hoc grounding.

2.3. LLM multi-agent orchestration

PentestAgent adopts RAG-grounded, role-based collaboration (reconnaissance, triage, exploitation) to reduce hallucinations and improve next-step selection [8]. Additionally, VulnBot structures collaboration via a Penetration Task Graph (PTG) to preserve phase order (recon \rightarrow scanning \rightarrow exploitation) and constrain branching; reported results include 30.3% overall and 69.05% subtask completion on AutoPenBench and strong performance on AI-Pentest-Benchmark [9, 10, 11]. RefPentester

[19] introduces knowledge-informed self-reflection tied to stage recognition, improving recovery from failed operations on Hack The Box targets. RapidPen targets the initial foothold (IP-toshell) with a ReAct-style loop and retrieval of exploit knowledge, demonstrating fully autonomous compromises on HTB within minutes at modest cost [20]. The work of Weber et al. [21] presents Perses, a notable effort to enable small language models (SLMs) to perform automated privilege escalation through an extensible, role-specialized multi-LLM architecture. Perses shows that heterogeneity—assigning lightweight models to Planner, Commander, Summariser and domain-specific Overseers—can substantially improve exploitation of simple misconfigurations. Importantly, the evaluation in Perses is narrowly scoped: experiments are conducted primarily on FreeBSD targets, employ a limited and largely handcrafted set of privilegeescalation vulnerabilities, and use a threat model tailored to configuration errors rather than broad end-to-end attacks. As a result, Perses demonstrates the viability of SLM heterogeneity in constrained environments but leaves open questions about transferability to full penetration pipelines (reconnaissance, scanning, multi-stage exploitation), complex real-world services, and heterogeneous network topologies.

2.4. Focused exploit studies (one-day, zero-day) and CTF-style agents.

Fang et al. [22] show that, given CVE descriptions, GPT-4 can exploit 87% of a 15 one-day vulnerability set, whereas other LLMs and scanners achieve 0%; without the description, success drops markedly (7%). Zhu et al. [23] extend to teams of agents (*HPTSA*) for *zero-day* web vulnerabilities, reporting up to 42% pass@5 and 18% pass@1 on 14 real-world cases with GPT-4. For broader skill evaluation, HackSynth proposes a two-module agent and two CTF-based benchmarks (PicoCT-F/OverTheWire; 200 tasks) [24], while NYU CTF Bench (NeurIPS D&B) contributes a scalable open-source dataset and automation framework (200 CSAW CTF tasks) [25].

2.5. Benchmarks and methodology

The emergence of AI-driven penetration testing has been accompanied by a rapid proliferation of evaluation suites designed to measure autonomy, tool integration, and end-to-end performance under controlled conditions. While the space remains nascent, a few benchmarks have begun to dominate experimental protocols. Notably, AutoPenBench and AI-Pentest-Benchmark appear most frequently in recent studies, reflecting their alignment with realistic, multi-phase pentesting workflows and their ability to grade performance across autonomy levels and subtasks. Conversely, more specialized testbeds such as CVE-Bench target specific exploitability dimensions (e.g., realworld CVEs in web contexts) and thus see adoption in works focusing on vulnerability exploitation rather than full-cycle orchestration. Capture-the-flag (CTF)-oriented resources such as NYU CTF Bench and the datasets introduced by HackSynth have also gained traction, particularly for skill-granular or task-decomposed evaluations, though their scenarios often differ from operational pentests in scope and realism.

Within this landscape, AutoPenBench offers open, graded tasks spanning web, network, and cryptographic targets, with configurable autonomy modes to support comparisons between orchestration strategies and model backbones [10]. AI-Pentest-Benchmark provides VM-based targets, enabling reproducible end-to-end penetration tests [11], thereby supporting performance attribution across discovery, exploitation, and post-exploitatioability scanning serve as data ingestion, exploitation is model phases. CVE-Bench grounds evaluation in real-world web CVEs, reporting typical success rates in the low teens even for state-ofthe-art agents, highlighting the gap between research prototypes and robust autonomy [26]. Methodological recommendations across these works increasingly emphasize standardizing budget constraints, clearly labeling autonomy levels, and reporting detailed error modes to prevent overestimation of capabilities

2.6. Positioning of our work

Relative to single-agent PentestGPT [7] and post-breach AutoAttacker [18], we retain a multi-agent/PTG discipline akin to PentestAgent/VulnBot [8, 9] but differ in three ways: (i) prioritizing mid-scale, open backbones for cost-effective, on-prem deployment; (ii) employing grey-box phase prompting to maintain phase continuity while limiting drift; and (iii) aligning evaluation with open substrates (AutoPenBench, AI-Pentest-Benchmark, and where applicable CVE-Bench) under fixed budgets and subtask breakdowns [10, 11, 26, 27].

2.7. Takeaways.

Outcomes hinge on (i) grounding quality (RAG/summaries/validators) and (ii) orchestration discipline (roles/PTG/reflection). Single-agent pipelines set baselines; role-structured multiagent systems improve reliability; reflective and IP-to-shell variants push autonomy at kill-chain ends; one-day/zero-day studies quantify limits of discovery vs. exploitation [22, 23]. Our approach emphasizes cost-effective, reproducible deployment with mid-scale open models, PTG structure, and grey-box prompts under open, reproducible protocols [10, 11, 26, 27].

3. Background

3.1. Automated Penetration Testing

Penetration testing aims to evaluate the security of a target system by simulating adversarial behavior across phases such as reconnaissance, vulnerability enumeration, exploitation, and privilege escalation. Manual execution is effective but limited by human resources and scalability. Automated Penetration Testing (APT) addresses these limitations by orchestrating these phases through intelligent agents and machine learning models.

Formally, let T denote a target system with configuration space C and attack surface S. An automated pentesting operation can be represented as a pipeline:

$$f: T \mapsto \{R, V, E, P\} \mapsto O$$

where R are the reconnaissance results (asset discovery, service mapping), V are detected vulnerabilities, E denotes the exploit simulation results, P represents the privilege escalation attempts and O is the structured output (reports, risk scores, or attack paths).

Such pipelines resemble MLOps workflows, where data collection, model inference, and result verification are continuously orchestrated. In this analogy, reconnaissance and vulnerinference, and reporting acts as the 'evaluation' stage. By automating this operation, APT enables repeatability, scalability, and integration into CI/CD security pipelines.

3.2. Multi-Agent AI Systems

Multi-Agent Systems (MAS) provide a natural architecture for automated penetration testing by assigning each pentest phase to a specialized agent. For example:

- Reconnaissance Agent: enumerates hosts, ports, and services (similar to VulnBot's 'Recon Agent' [9]).
- Vulnerability Analysis Agent: correlated scan data with CVE / CWE knowledge bases.
- Exploitation Agent: generates and tests candidate pay-
- Reporting Agent: summarizes results, attack graphs, and remediation advice.

Agents communicate through a task manager or memory module, allowing modularity and fault tolerance. Frameworks like CAMEL [29] show that role-conditioned LLM agents can collaborate effectively on complex objectives. In our system xOffense, MAS design ensures that each offensive task is handled by a role-specialized model while maintaining global coordination.

3.3. LLM-based Offensive Agents

Within MAS, LLMs provide reasoning, contextual understanding, and code synthesis capabilities that align well with penetration testing workflows. The central problem is, given an attack context C (system description, logs, CVEs), generate actionable steps or payloads A that maximize the likelihood of successful exploitation:

$$g: C \mapsto A$$

Although proprietary LLMs (e.g., GPT-4, Claude) offer strong performance, they introduce limitations in cost, reproducibility, and security control. Therefore, we adopt open-source models such as Qwen3, which support local deployment, fine-tuning, and quantization (AWQ/INT4), making them suitable for offensive research environments where sensitive data cannot leave the infrastructure.

To mitigate risks such as hallucination or unsafe outputs, LLM agents are sandboxed and validated against controlled execution environments before results are accepted.

Table 1: Comparison of representative automated pentesting systems and benchmarks (grid-lined). Criteria emphasize architecture, scope, grounding, tools, autonomy, and evaluation.

Work	Architecture (Arch.)	Scope/Phase	Grounding/ Memory	Tool Use	Evaluation & Highlights
PentestGPT [7]	SA	Web+Net (multi-phase)	SUM (module summaries)	Parse scans → cmd synth.	USENIX'24 cases/bench; modular pipeline
AutoAttacker [18]	SA (post)	Post-breach, OpSec realism	CTX (in-session)	Shell, Metasploit	Simulated org (Win/Linux); multi-step attacks
PentestAgent [8]	MA	Web focus (ext. assess.)	RAG + MEM	Scanners, PoCs	Bench + HTB; reduced hallucinations via RAG
VulnBot [9]	MA + PTG	Full cycle (recon→exp.)	Phase SUM (PTG state)	Nmap, Nikto, Metasploit	AutoPenBench (30.3% overall; 69.05% subtask), AIPB best-of-six
RefPentester [19]	MA + RFL	Stage-aware triage/exp.	Reflection + knowledge	Std. toolchain	HTB "Sau": +16.7% vs GPT-4o baseline
Perses [21]	MA (multi-LLM)	Privilege escalation	HET (heterogeneous model/task)	Tool-grounded (details in paper)	FreeBSD systems; small-LLM focus
RapidPen [20]	SA	IP→Shell (initial foothold)	MEM + exploit retrieval	Scan→exploit loop	HTB: autonomous shells in minutes; low cost
One-Day Agent [22]	SA	Web (one-day CVEs)	CVE-guided CTX	Browser, tools	87% (GPT-4, with CVE desc.); 7% w/o desc.
HPTSA (Teams) [23]	MA (hier./team)	Web (zero-day)	Planner + experts	Browser, task agents	42% pass@5; 18% pass@1 on 14 real vulns
HackSynth [24]	SA (2-mod.)	CTF (200 tasks)	Planner + summarizer	Sandbox tools	PicoCTF/OTW benchmarks; GPT-4o best
AutoPentest [28]	MA (LangChain)	Black-box (enum→exp.)	Prompting + MEM	Std. scans/exploits	GPT-4o-based prototype; open code
CVE-Bench [26]	Bench	Real web CVEs	_	-	Up to 13% s-rates for SOTA agents
AutoPenBench [10]	Bench	Mixed (Web/Net/CRPT)	_	_	33 tasks; autonomy and milestone scoring
NYU CTF Bench [25]	Bench	CTF (CSAW; 200)	-	Tool-integrated	NeurIPS D&B open dataset+automation
Our work	MA + PTG	Full cycle (Web+Net)	GBP + MEM	Broad scan/exploit	APB, AIPB, (opt.) CVEB; mid-scale open LLM

Legend: SA=Single-agent; MA=Multi-agent; PTG=Penetration Task Graph; RFL=Reflection; SUM=Summaries; RAG=Retrieval-Augmented Generation; MEM=Explicit memory; CTX=In-session context; HET=Heterogeneity (multi-LLM model/task selection); GBP=Grey-box phase prompting; APB=AutoPenBench [10]; AIPB=AI-Pentest-Benchmark [11]; CVEB=CVE-Bench [26]. "s-rate" denotes success rate.

3.4. Fine-tuning Methods for Offensive LLMs

Adapting LLMs to offensive security tasks requires specialization beyond general pre-trained knowledge, and several parameter-efficient fine-tuning approaches have been explored, including Prefix-Tuning, Adapter-based methods, Low-Rank Adaptation (LoRA) and its extension QLoRA.

Prefix-Tuning or P-Tuning v2 appends task-specific continuous vectors or tokens to the input prompt. Its advantages lie in simplicity and a very low memory footprint, since only the prefix embeddings are optimized. However, this method often struggles to capture deeper structural knowledge, and its effectiveness diminishes when reasoning requires multi-step tool interaction or long-context planning, both of which are essential in penetration testing.

Adapter-based methods insert small trainable modules within transformer layers, enabling modular adaptation to new tasks. They provide good task isolation and make it possible to reuse the same backbone across different domains with minimal additional parameters. Nonetheless, these methods introduce extra inference latency due to the added modules, and their limited capacity makes them less effective for embedding highly domain-specific procedural knowledge such as exploit reasoning or vulnerability chaining.

LoRA and its extension QLoRA decompose weight updates into low-rank matrices, significantly reducing the number of trainable parameters while retaining the expressive power of the base model. LoRA achieves a balance between efficiency and performance: it requires far fewer resources than full finetuning, adds negligible inference overhead, and can effectively embed specialized knowledge without erasing the general reasoning ability of the original model. Conceptually, weight updates ΔW lie in a low-rank subspace:

$$\Delta W = AB$$
, $A \in \mathbb{R}^{d \times r}$, $B \in \mathbb{R}^{r \times k}$, $r \ll \min(d, k)$

and the effective weight during inference is

$$W' = W + \alpha AB$$

where α is a scaling factor controlling the magnitude of the adaptation.

Considering the need to adapt a 32B-parameter model such as Qwen3 under realistic hardware constraints, this work adopts LoRA fine-tuning. This approach makes it possible to embed offensive knowledge—covering vulnerability patterns, exploit reasoning, and payload generation—efficiently while preserving the base model's general-purpose capabilities. Detailed dataset construction and the training procedure are described in Section 4.

4. Methodology

4.1. Overview of the proposed framework

xOffense is an innovative, lightweight framework for autonomous penetration testing, engineered to replicate the collaborative dynamics of human security teams while operating within resource-constrained environments. By harnessing compact Large Language Models (LLMs) with approximately 32 billion parameters, xOffense eliminates dependency on commercial APIs, enabling deployment on standard hardware. The framework decomposes the intricate process of penetration testing into three meticulously designed phases—reconnaissance, scanning, and exploitation - coordinated through a sophisticated multi-agent architecture. Comprising five core components -Task Orchestrator, Knowledge Repository, Command Synthesizer, Action Executor, and Information Aggregator - xOffense ensures seamless task progression, robust information management, and precise execution. This section elucidates the system's architecture, role delineation, task coordination, interagent communication, and execution mechanisms, underscoring its efficacy in addressing cybersecurity challenges with minimal computational overhead.

The operational workflow of *xOffense*, illustrated in Figure 1, initiates when a user submits a penetration testing objective, such as "Identify vulnerabilities on IP 192.168.X.X and retrieve

root-level flags." This task description serves as the Initial Context, which is passed to the *Task Orchestrator* for comprehensive plan generation. The orchestrator constructs a Task Coordination Graph (TCG), decomposing the penetration objective into a structured sequence of tasks with clearly defined dependencies. To enhance contextual accuracy, it queries the *Knowledge Repository*—a vector-based database—via a Retrieval - Augmented Generation (RAG) mechanism, which is taken from Langchain-Chatchat [30], retrieving relevant penetration knowledge based on inputs such as the initial task description, the current task's instruction, or recent task results.

Each task within the TCG is processed through an iterative loop involving Command Synthesis, Execution, Feedback Analysis, and Dynamic Plan Update. The Command Synthesizer, fine-tuned using lightweight LoRA techniques, translates task directives into precise, tool-specific commands, which are executed by the Action Executor utilizing a MemAgent - enhanced context management system to handle verbose outputs effectively. Post-execution, task outcomes are evaluated and relayed back to the orchestrator, which marks tasks as completed or triggers reflection and re-planning in case of failures. Upon completing all tasks within a phase (e.g., Reconnaissance), the Information Aggregator consolidates outputs into concise directives for the subsequent phase (e.g., Scanning), ensuring contextual coherence and minimizing token overhead. This orchestration - execution-feedback loop is consistently applied across all three phases, with each phase iteratively refining the attack path based on task outcomes and environmental feedback. The overall workflow, including execution and re-planning, is later formalized in Algorithm 1. The workflow terminates upon successfully achieving the defined success criteria, such as privilege escalation or flag retrieval.

4.2. Role Specialization

To navigate the complexity of penetration testing, xOffense employs a role specialization strategy, mitigating the risk of information overload and ensuring contextual coherence across phases. By assigning agents to distinct roles, the framework optimizes resource utilization and maintains precision in task execution, addressing the challenge of dynamic reasoning across testing stages.

- **Reconnaissance Phase**: Agents focus on comprehensive intelligence gathering, cataloging network configurations, open ports, and service details. Tools such as *Nmap* [13] for network scanning, *Dirb* [31] for directory enumeration, *Gobuster* [32] for brute-forcing hidden directories, and *Amass* [33] for subdomain discovery are integrated. For instance, a task might execute nmap ¬sV ¬p¬ <target-ip> to map all open ports and services, providing a robust foundation for subsequent phases.
- Scanning Phase: Building on reconnaissance insights, scanning agents identify vulnerabilities and misconfigurations using tools like *Nikto* [14] for web server analysis, WPScan [15] for WordPress vulnerabilities, sqlmap [34] for SQL injection testing for comprehensive vulnerability

- scanning. This phase prioritizes exploitable weaknesses to streamline progression.
- Exploitation Phase: Agents exploit identified vulnerabilities to gain unauthorized access or escalate privileges, employing tools such as *Metasploit* [4] for exploit development, *Hydra* [35] for credential brute-forcing, *John the Ripper* [36] for password cracking, and *ExploitDB* [37] for sourcing exploit scripts. For example, a task might deploy a Metasploit module to exploit a known CVE, followed by privilege escalation via a custom script.

This structured delineation ensures that each phase leverages prior findings, fostering a cohesive testing process and mitigating the risk of fragmented analyses.

4.3. Task Coordination and Reflection

The Task Coordination Graph (TCG) and its integrated *Check and Reflection Mechanism* form the cornerstone of xOffense's penetration path planning, enabling systematic task execution and adaptive plan refinement. These components address the challenges of limited context windows and inadequate error handling, ensuring robust and dynamic testing workflows. Their operational logic is illustrated in Algorithm 1, Algorithm 2, and Algorithm 3.

4.3.1. Task Coordination Graph

The TCG is a structured acyclic digraph, defined as G = (V, E), where V represents individual tasks and E denotes dependencies, ensuring logical and conflict-free execution. Each task node $v \in V$ encapsulates attributes such as:

- Directive: A clear instruction, e.g., "enumerate services on port 80 of 192.168.X.X."
- Operation Type: Specifies whether the task involves automated shell commands (e.g., nmap) or manual intervention.
- Prerequisites: Lists tasks that must be completed prior to execution, ensuring sequential integrity.
- Command: The tool-specific instruction generated by the *Command Synthesizer*.
- Outcome: The execution result, capturing tool outputs or errors.
- Completion Status: Indicates whether the task is completed or pending.
- Success Status: Records whether the task was successful.

The *Task Orchestrator* generates the TCG in a JSON - compliant format, dynamically updating it based on execution outcomes. For instance, a task designed to perform user authentication by initiating an SSH connection attempt—targeting a specific service endpoint associated with remote shell access protocols—on port 22 of 192.168.X.X depends on the successful completion of a preceding port scanning task. Subsequent

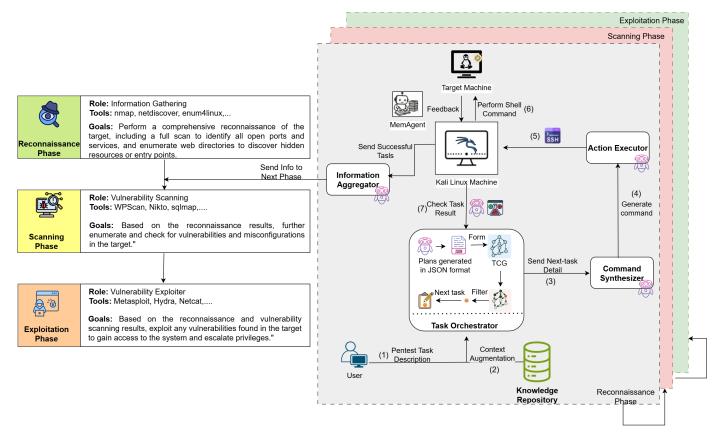


Figure 1: The Overall Architecture of the xOffense Framework.

tasks, such as performing an exhaustive enumeration of writable directories for privilege escalation through misconfigured permissions or publicly writable paths — (find / -writable 2>/dev/null) or listing running processes (ps aux), are contingent on this authentication.

Figure 2 illustrates a sample TCG, with a JSON task list on the left detailing directives, dependencies, and commands, and a dependency graph on the right showing task sequences with arrows indicating prerequisites. This formalism provides the structural foundation later used in Algorithm 1 for iterative execution and feedback handling.

The TCG operates through two sessions:

- **Planning Session**: The *Task Orchestrator* constructs an initial action plan tailored to the target system's characteristics and user requirements. It decomposes the plan into structured task lists, ensuring logical sequencing and dependency alignment. The plan is dynamically refined based on execution feedback, addressing the challenge of maintaining coherent context across phases.
- Task Session: This session generates detailed instructions for each task, which are passed to the *Command Synthesizer* for command generation and to the *Action Executor* for execution. It also evaluates execution outcomes, updating the TCG's completion and success statuses, as shown in Algorithm 1.

This structured approach ensures systematic progression, mitigating the risk of out-of-sequence execution and enhancing efficiency in resource-constrained environments.

4.3.2. Check and Reflection Mechanism

The generation of erroneous commands and the lack of effective error-handling mechanisms pose significant challenges to LLM-based penetration testing. xOffense addresses these issues through a *Check and Reflection Mechanism* integrated into the *Task Orchestrator*, enabling continuous self-assessment and plan optimization. The full workflow is detailed in Algorithm 1.

During the *Task Session*, the *Action Executor* evaluates task outcomes and updates the *Task Coordination Graph (TCG)* with success or failure statuses. The *Planning Session* then reflects on these outcomes, revising task directives and updating the TCG accordingly. Successful tasks are retained, while failed tasks trigger a reanalysis process, wherein the LLM regenerates commands with corrected parameters or alternative strategies. The updated plan is merged with previously completed tasks to preserve execution continuity, as demonstrated in Algorithm 2 and Algorithm 3.

Figure 2: Task Coordination Graph (TCG) illustrating task dependencies and execution status. Completed tasks are shown in dark, the current task in orange, and pending tasks in light blue.

Algorithm 1 Check and Reflection Procedure

```
Require: TCG (Task Coordination Graph), Knowledge Repository KR
```

```
1:
    while not all tasks completed do
        t \leftarrow \text{NextTask}(\text{TCG})
2:
        r \leftarrow \texttt{Execute}(t)
 3:
 4:
        if CheckSuccess(r) then
 5:
             MarkCompleted(t)
             StoreEmbedding(t, r, KR)
 6:
 7:
        else
 8:
             K \leftarrow \text{RetrieveSimilar}(t, KR)
 9:
             t' \leftarrow \text{RegenerateTask}(t, K)
             MergeTasks(t', TCG)
10:
11:
        end if
        UpdatePlan(TCG)
12:
13: end while
```

Plan Update and Merge Algorithms.. The UpdatePlan and MergeTasks procedures are key to preserving execution continuity. Upon task failure, the system calls the LLM to propose an updated plan, then merges it with the existing TCG such that all successfully completed tasks are retained, sequence numbers are adjusted, and only pending or failed tasks are revised. Their formal pseudocode is shown in Algorithm 2 and Algorithm 3.

Algorithm 2 UpdatePlan: LLM-driven Plan Revision

```
Require: Current plan P, failed task t, result r

1: S \leftarrow list of completed-success tasks from P

2: F \leftarrow list of failed tasks from P

3: P_{\text{new}} \leftarrow \text{LLMUPDATEPLAN}(t, r, S, F)

4: P^* \leftarrow \text{MergeTasks}(P, P_{\text{new}})

5: return P^*
```

Algorithm 3 MergeTasks: Success-Preserving Integration

```
Require: Old plan P_{\text{old}}, new plan P_{\text{new}}
 1: C \leftarrow completed-success tasks from P_{\text{old}}
 2: \mathcal{M} \leftarrow empty list
 3: for all \tau \in C do
          if \tau not in P_{\text{new}} by instruction then
 4:
 5:
               append \tau to \mathcal{M} with reset dependencies
 6:
          end if
    end for
 7:
    for all \hat{\tau} \in P_{\text{new}} do
 8:
          if instruction matches a task in C then
 9:
10:
               reuse completed task with updated dependencies
11:
               append \hat{\tau} as a new task
12:
          end if
13:
14: end for
15: update sequence numbers in \mathcal{M}
16: return merged plan with tasks \mathcal{M}
```

The *Knowledge Repository* supports this mechanism indirectly by assisting the *Task Orchestrator* during plan updates. It stores embeddings of previously successful tasks and curated

penetration testing knowledge (e.g., exploitation techniques, privilege escalation methods, and tool usage tutorials from sources such as HackTricks [38] and HackingArticles [39]). When replanning, the *Task Orchestrator* queries this repository to retrieve the top-k most relevant past cases using vector similarity search. Retrieved results are re-ranked and integrated into the revised TCG, ensuring that updates benefit from prior successes. This integration enhances resilience against hallucinated commands, improves error recovery, and maintains efficiency across iterative penetration testing phases.

4.4. Inter-Agent Communication Mechanism

Seamless coordination among agents is critical for maintaining contextual coherence across the reconnaissance, scanning, and exploitation phases, particularly given the limited context window of compact LLMs. xOffense employs the *Information Aggregator* to facilitate efficient communication, consolidating verbose outputs into concise, actionable summaries to optimize token usage and prevent information overload. The entire communication pipeline is operationalized in Algorithm 4.

For example, reconnaissance outputs—such as open ports (e.g., 22, 80, 443), service versions, and system fingerprints—are synthesized into a compact directive for the scanning phase, enabling targeted vulnerability detection with tools like *Nikto* or *sqlmap*. Similarly, scanning outputs, such as a SQL injection vulnerability identified by *sqlmap* or a misconfiguration detected by *Nuclei*, are summarized to guide the exploitation phase in prioritizing relevant exploits.

The *Information Aggregator* maintains a persistent shell state log, tracking access levels (e.g., a low-privileged user account gained via SSH) and system context (e.g., operating system type). This log ensures continuity across phases, mitigating the risk of context loss and enabling dynamic integration of findings. By filtering outputs from preceding phases to focus on critical insights, the mechanism minimizes computational overhead, ensuring efficient operation on a 32B-parameter LLM. This streamlined communication fosters a cohesive testing process, addressing the challenge of synthesizing information across multiple stages.

4.5. Generative Behavior and Execution

xOffense supports three operational modes—automatic, semi-automatic, and manual—with the automatic mode enabling fully autonomous testing. The *Command Synthesizer* transforms TCG directives into precise, tool-specific instructions tailored to the target system and phase, addressing the challenge of accurate command generation.

For instance, a reconnaissance directive might yield nmap -sS -p 22,80,443 <target-ip> for stealth scanning, while a scanning task might produce sqlmap -u http://<target-ip>/login --batch to test for SQL injection vulnerabilities or nikto -h http://<target-ip> for web server analysis. In the exploitation phase, commands like use exploit/windows/smb/ms17_010_eternalblue in *Metasploit*.

The Action Executor runs these commands via a Python Paramiko-based interactive shell on a Kali Linux environment,

Algorithm 4 Inter-Agent Communication via PlannerSummary

```
1: Input: Phase sequence P = \{p_1, p_2, \dots, p_n\}, Shell State Log S
 2: for i = 1 to n - 1 do
       // Step 1: Collect and summarize results from previous
   phase
       history_ids \leftarrow GetPlannerIDs(p_i)
 4:
 5:
       if |\text{history\_ids}| = 0 then
 6:
           context \leftarrow "
 7:
 8:
           summary ← "Previous Phase:\n"
9:
           for each id in history_ids do
10:
              plan \leftarrow \texttt{get\_planner\_by\_id}(id)
11:
              for each task in plan.finished_tasks do
                  summary ← summary || "Instruction:
12:
    + task.instruction
                           " + task.code
         + || "Code:
            || "Result: " + task.result
           "\n----\n"
              end for
14:
           end for
15:
           context, _ ← callLLM(query = write_summary +
    summary, summary = False)
16:
       end if
       // Step 2: Send summarized context to next phase planner
17:
       InitPlanner(p_{i+1}, context = context, state = S)
18:
19: end for
```

simulating human interactions with high fidelity. This component seamlessly processes tool-specific instructions generated by the Command Synthesizer, enabling robust interaction with the target system through simulated keyboard operations. The Action Executor is optimized for the **Qwen3-32B** model, which, despite its constrained 16,384-token context window, integrates the innovative MemAgent [40] framework to handle extended contexts effectively. Drawing on MemAgent's segment-based processing and reinforcement learning (RL)optimized memory mechanism, as described in the referenced study, the Action Executor processes arbitrarily long outputs by iteratively reading command results in chunks and updating a fixed-length memory. This approach ensures linear computational complexity, allowing xOffense to manage verbose tool outputs without performance degradation, even beyond the Owen3-32B's native context limit. To address the challenge of excessive or redundant output, a sophisticated filtering mechanism employs the MemAgent-enhanced LLM to extract critical information when results exceed 8,000 characters, preserving only actionable insights for analysis. These insights are relayed to the Task Orchestrator for further processing, ensuring contextual coherence and minimizing computational overhead. By incorporating MemAgent's ability to selectively retain relevant data while discarding distractors, the Action Executor enhances the system's efficiency and scalability, enabling robust penetration testing in resource-constrained environments and contributing to xOffense's capability to handle complex, long-context workflows with precision.

5. Implementation, Benchmark Dataset and Metrics

To thoroughly assess the effectiveness and practicality of xOffense, we designed a comprehensive experimental setup that evaluates not only task completion rates but also the scalability, adaptability, and efficiency of the system under realistic penetration testing conditions. This section details the experimental settings, models, fine-tuning methodologies, benchmark datasets, evaluation metrics and presents an in-depth analysis of the empirical results.

5.1. Experimental Setup

5.1.1. Environment

Attacker Environment. All penetration testing experiments were executed from a dedicated attacker machine configured as a VMware virtual workstation running Kali Linux 2025 [41]. Kali was chosen for its comprehensive penetration testing toolkit and compatibility with industry-standard workflows. This virtualized attacker host was provisioned with 8 vCPUs, 16 GB RAM, and 120 GB storage, ensuring stable execution of both offensive tools and the *xOffense* multi-agent framework within a single environment.

Victim Environment. Two types of target environments were deployed corresponding to the benchmark datasets:

- AUTO-PEN-BENCH [10]: Tasks were instantiated as
 Docker containers, which were hosted on a separate vir tual machine to avoid resource contention with the at tacker host. This victim VM was configured with 4 vC PUs, 8 GB RAM, and 80 GB storage, and placed in the
 same NAT network as the attacker machine to ensure di rect connectivity.
- AI-Pentest-Benchmark [11]: Vulnerable machines were directly imported from official VulnHub distributions and executed as VMware virtual machines without modification to their default specifications, in order to preserve the original exploitation conditions. All machines were assigned to the same NAT network as the attacker host to guarantee consistent communication.

5.1.2. Model Serving

The fine-tuned model, namely Qwen3-32B-finetune was hosted on a dedicated compute node equipped with an NVIDIA A100 GPU (80 GB VRAM). The same hardware was also used during the fine-tuning process to accelerate training efficiency. For inference, the model was exposed via an API endpoint tunneled through ngrok, allowing the attacker machine to interact with the model as an external service. This separation ensured that LLM inference did not compete with penetration testing tasks for system resources, while also replicating realistic deployment conditions where models are often served remotely.

5.1.3. Evaluated Models

The evaluation considered both commercial and open-source large language models to provide a comprehensive baseline. Specifically, we included GPT-40, Llama3.3-70B, Llama3.1-405B, and DeepSeek-V3. For open-source mid-scale models, we assessed Qwen3-32B-base, which provides a strong balance between efficiency and capability. Building upon this foundation, we fine-tuned the model into Qwen3-32B-finetune, specialized for penetration testing tasks within *xOffense*. All models were evaluated with a temperature setting of 0.5 to balance deterministic output with generative flexibility, ensuring consistency in automated penetration testing scenarios.

5.2. Fine-Tuning Methodology

Unlike conventional large-scale model adaptations, which demand significant computational overhead, *xOffense* leverages LoRA (Low-Rank Adaptation) to achieve domain specialization. LoRA reduces the parameter footprint by freezing the base model's weights and training only a compact set of adapter matrices. This approach dramatically lowers the number of trainable parameters by over 99%, enabling efficient fine-tuning even on standard GPU infrastructures.

To further address the memory bottlenecks of handling a 32B-parameter model, we employ DeepSpeed ZeRO-3 [42] optimization. ZeRO-3 partitions model states—including optimizer, gradients, and parameters—across multiple GPUs, achieving linear scalability. Additionally, FlashAttention v2 [43] is integrated to optimize attention computation, reducing memory usage and accelerating training by up to 3x compared to standard attention implementations. These combined techniques allow us to efficiently fine-tune Qwen3-32B for penetration testing workloads with a significant reduction in hardware demands. And the fine-tuning dataset comprised two main corpora as follows:

• PentestData: This dataset is meticulously curated to encompass domain-specific question-answer pairs, each enriched with synthetically generated Chain-of-Thought (CoT) reasoning traces. The reasoning steps are encapsulated within <think> tags, enabling the model to learn structured, step-by-step logical deduction processes tailored for penetration testing scenarios. To construct PentestData, we aggregate and standardize write-ups from over 1,000 machines across leading cybersecurity platforms, including TryHackMe [44], HackTheBox [45], and VulnHub [46]. These write-ups are systematically processed to extract task-specific reasoning paths, exploit procedures, and decision-making sequences relevant to offensive security operations. In addition, we incorporate supplementary pentest datasets from HuggingFace Datasets Hub [47], focusing on cybersecurity knowledge bases, penetration testing techniques, and practical guides for commonly used security tools. This comprehensive integration ensures that PentestData serves as a robust and diverse resource for training models in autonomous penetration testing workflows.

• WhiteRabbitNeo: A high-quality JSONL-formatted dataset comprising instruction-response pairs, specifically curated for cybersecurity tasks. Although the original dataset lacked explicit Chain-of-Thought (CoT) reasoning annotations, it was standardized during preprocessing by appending empty <think> tags to each sample. This structural unification ensures compatibility with CoT-augmented training pipelines and facilitates subsequent fine-tuning for step-by-step reasoning abilities. The dataset draws from real-world offensive and defensive cybersecurity scenarios, encompassing exploitation techniques, payload crafting, and red-team/blue-team interactions, sourced from the WhiteRabbitNeo community contributions [48].

5.3. Benchmark Datasets

To ensure a rigorous and practically relevant evaluation, we selected two complementary benchmarks that cover both synthetic vulnerabilities and realistic multi-stage exploitation scenarios. Together, these benchmarks provide a balanced testbed for assessing the adaptability and robustness of automated penetration testing systems.

AutoPenBench. This benchmark defines a total of 33 penetration testing tasks, spanning both instructional "in-vitro" exercises and real-world CVEs. The in-vitro set (22 tasks) reflects fundamental vulnerability classes frequently highlighted in industry rankings such as the OWASP Top 10, including weak access control (e.g., misconfigured sudo, world-writable shadow files), web application flaws (e.g., path traversal, SQL injection, file upload RCE), and insecure network configurations (e.g., SNMP misconfiguration, ARP spoofing). In addition, four cryptography tasks evaluate resilience against improper or weak cryptographic implementations, such as bruteforcing Diffie-Hellman keys. Beyond these educational tasks, the benchmark incorporates 11 real-world CVEs ranging from 2014 to 2024 with CVSS scores between 7.5 and 10.0. These include critical vulnerabilities widely recognized for their impact and prevalence, such as Log4Shell (CVE-2021-44228), Heartbleed (CVE-2014-0160), SambaCry (CVE-2017-7494), and Spring4Shell (CVE-2022-22965). By combining foundational categories with critical CVEs, AUTOPENBENCH provides a structured yet realistic environment to evaluate whether agents can transition from basic exploitation to handling highseverity vulnerabilities that have historically dominated realworld attack campaigns.

AI-Pentest-Benchmark.. While AutoPenBench focuses on containerized vulnerabilities, the AI-Pentest-Benchmark evaluates AI agents on complete end-to-end exploitation workflows across 13 real-world vulnerable machines drawn from VulnHub. These machines are categorized by difficulty into easy (e.g., Victim1, Library2, Funbox, WestWild), medium (e.g., Cengbox2, Devguru, Symfonos2), and hard (e.g., Insanity, TempusFugit). Each machine defines a structured set of reconnaissance, exploitation, privilege escalation, and general technique subtasks, amounting to 152 tasks in total. The

vulnerabilities embedded within these machines reflect common penetration testing scenarios, including web application flaws (SQL injection, XSS, CSRF/SSRF), network service weaknesses (FTP/AD enumeration, brute-force authentication), code-level issues (deserialization, command injection), and post-exploitation techniques (cronjob analysis, misconfigured system files, privilege escalation via user access exploitation). Notably, many of these tasks map directly to recurring weakness categories in the CWE Top 25 and OWASP Top 10, ensuring that success on this benchmark corresponds to capabilities relevant in practical offensive security operations. The benchmark is particularly challenging because the ultimate goal is to achieve root access on each machine, requiring coherent reasoning across reconnaissance, exploitation, and privilege escalation stages. Previous studies have shown that even large-scale proprietary models such as GPT-4o and Llama3.1-405B were unable to achieve root-level compromise without human assistance, underscoring the difficulty and realism of this bench-

5.4. Evaluation Metrics

To evaluate the performance of *xOffense*, we employ three complementary metrics that capture both high-level task success and fine-grained sub-task robustness.

5.4.1. Overall Task Completion Rate

This metric measures the percentage of target machines successfully compromised (i.e., the flag was obtained) within the allowed interaction budget. Formally:

Overall Rate =
$$\frac{\text{\# compromised machines}}{\text{\# total machines}}$$

This provides a coarse-grained view of whether an agent can achieve end-to-end exploitation across categories such as Access Control (AC), Web Security (WS), Network Security (NS), Cryptography (CRPT), and Real-world tasks.

5.4.2. Sub-task Completion Rate (1 Experiment)

To gain insight into intermediate stages of penetration testing, we evaluate sub-task success rates (e.g., service enumeration, vulnerability detection). Each benchmark defines a set of subtasks *S*. A sub-task is considered successful if it is completed in at least one of the five independent runs:

Subtask-1Exp =
$$\frac{|\{s \in S \mid \exists i \in [1, 5], \text{ success}(s, i)\}|}{|S|}$$

This metric highlights the agent's ability to eventually solve a sub-task, even if not consistently across all runs.

5.4.3. Sub-task Completion Rate (5 Experiments)

To measure robustness and consistency, we also compute the cumulative completion rate across all five runs. In this case, we count the total number of successful sub-tasks over all experiments and normalize by the maximum possible number of successes:

Subtask-5Exp =
$$\frac{\sum_{i=1}^{5} \text{ successes}(i)}{5 \times |S|}$$

This stricter metric rewards agents that not only succeed once but can repeatedly complete subtasks across independent executions.

Together, these three metrics provide a balanced view: (i) overall penetration capability, (ii) eventual solvability of subtasks, and (iii) robustness of performance under repeated trials.

6. Evaluation and results

6.1. Experiment Scenarios

We design five experimental scenarios to comprehensively evaluate *xOffense* across synthetic and real-world settings:

- Scenario 1: Overall task completion on AutoPenBench, measuring full machine compromise across AC, WS, NS, and CRPT categories.
- Scenario 2: Sub-task completion on AutoPenBench (1 Experiment), where a sub-task is successful if solved in at least one of five runs.
- Scenario 3: Sub-task completion on AutoPenBench (5 Experiments), aggregating successful subtasks across all five runs to capture consistency.
- Scenario 4: Real-world exploitation on AI-Pentest-Benchmark without RAG, using six representative VulnHub machines.
- Scenario 5: Real-world exploitation on AI-Pentest-Benchmark with RAG, highlighting the contribution of retrieval to complex exploitation chains.

6.2. Task Completion Performance Across Penetration Testing Categories

6.2.1. Overall Task Completion Performance

Table 2 presents the overall task completion rates across all evaluated models on the AUTO-PEN-BENCH dataset. The fine-tuned **Qwen3-32B-finetune** model achieved a remarkable **72.72**% completion rate, substantially outperforming both its base variant, **Qwen3-32B-base** (30.30%), and other state-of-the-art models, including GPT-4o (21.21%), Llama3.1-405B (Paper) (30.30%), and PentestGPT (9.09%).

The performance disparity between **Qwen3-32B-finetune** and its base version is particularly noteworthy. Despite having identical model architecture and parameter size (32 billion parameters), the domain-specific fine-tuning enabled a **2.4x improvement** in task completion. This validates the effectiveness of our lightweight LoRA fine-tuning pipeline in adapting general-purpose models to specialized penetration testing workflows.

In the Access Control (AC) category, Qwen3-32B-finetune achieved a 100% success rate, a stark contrast to the 40.00% of Qwen3-32B-base and the 60.00% of Llama3.1-405B (Paper). Similarly, in Network Security (NS), our model achieved 83.33% completion, surpassing all baselines, including Llama3.3-70B (33.33%) and Qwen3-32B-base (50.00%).

Notably, even in the complex **Real-world** category, **Qwen3-32B-finetune** attained a **54.54**% success rate, outperforming Qwen3-32B-base (27.27%) and PentestGPT (0.00%).

These findings demonstrate that fine-tuning on domain-relevant CoT data and incorporating robust task orchestration mechanisms can enable a quantized, resource-efficient model to match—and even surpass—the capabilities of larger, general-purpose LLMs in specialized scenarios. The consistent outperformance across categories further validates the robustness of our fine-tuning strategy, especially given the compute-efficient AWO quantization.

6.2.2. Sub-task Completion Performance (1 Experiment)

To assess finer-grained capabilities, we evaluated sub-task completion in a single-run experiment (Table 3). The term "1 Experiment" refers to the overall subtask completion rate across five experiments, where a subtask is considered successful if it succeeds in at least one experiment. **Qwen3-32B-finetune** achieved a **79.17**% sub-task completion rate, outperforming Llama3.1-405B (Paper) (69.05%) by a margin of **10.12**%. This margin is significant, particularly when considering that Llama3.1-405B is a much larger model (405B parameters) operating in its native configuration.

In the **Real-world** category, **Qwen3-32B-finetune** achieved a **35.96**% sub-task completion rate, more than doubling that of Qwen3-32B-base (24.92%) and outperforming Llama3.1-405B (Paper) (26.19%). Similarly, in the CRPT category, the fine-tuned model demonstrated a **3.41**% improvement over Llama3.1-405B (Paper).

Interestingly, while Qwen3-32B-base achieved moderate results (52.36%), its performance gap to Qwen3-32B-finetune (79.17%) illustrates the critical role of domain adaptation. The base model, though capable of handling general security tasks, struggled with multi-step reasoning and contextual coherence, particularly in chained exploit scenarios. The finetuned model's superior performance confirms that its CoT-driven prompt alignment and RAG-assisted knowledge retrieval mechanisms provide a tangible advantage in executing complex task sequences.

6.2.3. Sub-task Completion Performance (5 Experiments)

To evaluate robustness and stability, we conducted aggregated experiments over five runs (Table 4). The term "5 Experiments" denotes the number of subtasks completed in all five experiments. **Qwen3-32B-finetune** maintained its lead with a **60.94**% sub-task completion rate, significantly outperforming Llama3.1-405B (Paper) (49.90%) and Qwen3-32B-base (23.03%). This robustness is critical in practical pentesting workflows, where variance due to environmental noise and complex task dependencies often degrades model performance.

In the Access Control (AC) category, Qwen3-32B-finetune achieved a remarkable 14.51%, which is 4.32% higher than Llama3.1-405B (Paper). For Web Security (WS), our model reached 10.91%, outperforming all baselines by a significant margin. Notably, even in categories where task chains are inherently volatile—such as Real-world—the

Table 2: Overall Task Completion Rate on Target Machines. Our fine-tuned model demonstrates superior performance, especially in AC, NS, and Real-world categories.

Category	GPT-40	Llama3.3-70B (VulnBot)	Llama3.1-405B (VulnBot)	Llama3.1-405B (PentestGPT)	Qwen3-32B (Base)	Qwen3-32B-finetune (Ours)
AC	1 (20.00%)	1 (20.00%)	3 (60.00%)	1 (20.00%)	2 (40.00%)	5 (100.00%)
WS	2 (28.57%)	1 (14.29%)	2 (28.57%)	0 (0.00%)	2 (28.57%)	5 (71.42%)
NS	3 (50.00%)	2 (33.33%)	2 (33.33%)	2 (33.33%)	3 (50.00%)	5 (83.33%)
CRPT	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	0 (0.00%)	3 (75.00%)
Real-world	1 (9.09%)	2 (18.18%)	3 (27.27%)	0 (0.00%)	3 (27.27%)	6 (54.54%)
ALL	7 (21.21%)	6 (18.18%)	10 (30.30%)	3 (9.09%)	10 (30.30%)	24 (72.72%)

Table 3: Sub-task Completion Rate (1 Experiment). Qwen3-32B-finetune shows the highest completion rate across all categories.

Category	Llama3.3-70B	Llama3.1-405B	Llama3.3-70B	Llama3.1-405B	Llama3.1-405B	Qwen3-32B	Qwen3-32B-finetune
	(VulnBot)	(VulnBot)	(Base)	(Base)	(PentestGPT)	(Base)	(Ours)
AC	25 (11.90%)	31 (14.76%)	16 (7.62%)	21 (10.00%)	20 (9.52%)	26 (8.20%)	46 (14.51%)
WS	24 (11.43%)	30 (14.29%)	22 (10.48%)	26 (12.38%)	18 (8.57%)	28 (8.83%)	38 (11.98%)
NS	12 (5.71%)	11 (5.24%)	10 (4.76%)	9 (4.29%)	6 (2.86%)	11 (3.47%)	15 (4.73%)
CRPT	15 (7.14%)	18 (8.57%)	17 (8.10%)	18 (8.57%)	12 (5.71%)	22 (6.94%)	38 (11.98%)
Real-world	49 (23.33%)	55 (26.19%)	29 (13.81%)	29 (13.81%)	28 (13.33%)	79 (24.92%)	114 (35.96%)
ALL	125 (59.52%)	145 (69.05%)	94 (44.76%)	103 (49.05%)	84 (40.00%)	166 (52.36%)	251 (79.17%)

Table 4: Sub-task Completion Rate (5 Experiments). Our model maintains a significant lead, demonstrating robustness and consistency.

Category	Llama3.3-70B	Llama3.1-405B	Llama3.3-70B	Llama3.1-405B	Llama3.1-405B	Qwen3-32B	Qwen3-32B-finetune
	(VulnBot)	(VulnBot)	(Base)	(Base)	(PentestGPT)	(Base)	(Ours)
AC	87 (8.29%)	107 (10.19%)	46 (4.38%)	61 (5.81%)	27 (2.57%)	60 (3.78%)	212 (14.51%)
WS	106 (10.10%)	116 (11.05%)	83 (7.90%)	66 (6.29%)	40 (3.81%)	70 (4.42%)	173 (10.91%)
NS	41 (3.90%)	40 (3.81%)	36 (3.43%)	22 (2.10%)	15 (1.43%)	10 (0.63%)	71 (4.67%)
CRPT	65 (6.19%)	75 (7.14%)	68 (6.48%)	44 (4.19%)	43 (4.10%)	70 (4.42%)	176 (11.10%)
Real-world	166 (15.81%)	186 (17.71%)	99 (9.43%)	67 (6.38%)	56 (5.33%)	155 (9.78%)	334 (21.07%)
ALL	465 (44.29%)	524 (49.90%)	332 (31.62%)	260 (24.76%)	181 (17.24%)	365 (23.03%)	966 (60.94%)

fine-tuned model achieved **21.07**%, compared to 17.71% for Llama3.1-405B (Paper) and only 9.78% for Qwen3-32B-base.

While a performance drop of approximately 18% from the single-experiment run to the aggregated runs was observed, this is expected due to increased task complexity and stochastic failures inherent in autonomous pentesting. Nevertheless, the fine-tuned model's consistency across these iterations underscores its robustness, particularly when contrasted with PentestGPT's 17.24% sub-task completion in the same setting.

6.2.4. Comparative Insights

A critical observation from these experiments is the disproportionate performance leap achieved through fine-tuning relative to model size. Despite being a 32B parameter model, **Qwen3-32B-finetune** consistently outperformed larger counterparts like Llama3.1-405B (405B parameters) across every evaluation metric. This validates our hypothesis that task orchestration, RAG-driven context augmentation, and parameter-efficient tuning techniques (LoRA + ZeRO-3 + FlashAttention) can bridge, and in specialized scenarios, exceed the performance gap traditionally associated with sheer model size.

Furthermore, the disparity between Qwen3-32B-base and Qwen3-32B-finetune exemplifies the inadequacy of using general-purpose LLMs in specialized pentesting workflows without domain adaptation. The base model, though archi-

tecturally identical, lacked the reasoning depth and contextcoherence required for intricate attack path planning, resulting in lower task and sub-task completion rates.

6.3. Evaluation on Complex Real-World Exploitation Chains

6.3.1. Performance without Retrieval-Augmented Generation (No RAG)

To assess the baseline capabilities of our proposed system *xOffense* in realistic offensive security scenarios, we conducted experiments on the same set of six real-world vulnerable machines as utilized in the VulnBot [9] evaluation: Victim1, Library2, Sar, WestWild, Symfonos2, and Funbox. This machine set—originally derived from the AI-Pentest-Benchmark—covers a diverse range of exploitation challenges, including misconfigurations, weak authentication, remote code execution, privilege escalation, and multi-step attack chains. By adopting this identical set, we ensure methodological consistency and enable a direct, fair comparison with prior work.

The experiments were conducted in a fully autonomous mode, without any human intervention or Retrieval-Augmented Generation (RAG) support. Each target machine was tested in five independent runs, and the reported performance represents the best subtask completion rate per machine, following the AI-Pentest-Benchmark scoring methodology. Figure 3 presents the comparative results across multiple mod-

els, including VulnBot-Llama3.1-405B, VulnBot-DeepSeekv3, their respective base models, and our proposed Qwen3-32B variants (base and finetuned).

The results reveal several noteworthy patterns. First, **Qwen3-32B-finetune** consistently surpasses its base counterpart across all six machines, with particularly significant improvements on *Victim1* (+0.55), *Library2* (+0.30), and *West-Wild* (+0.63). These gains highlight the effectiveness of domain-specific fine-tuning in strengthening the model's exploitation reasoning and procedural robustness. Second, while VulnBot-DeepSeek-v3 remains highly competitive—achieving the highest score on *Victim1* (0.83) and *WestWild* (0.71)—our fine-tuned Qwen3-32B achieves comparable or superior performance on most other machines, including leading results on *Sar* (0.58) and *Funbox* (0.54).

Notably, performance disparities are strongly correlated with the complexity of exploitation chains. Targets such as *Symfonos2* and *Funbox*, which demand multi-stage privilege escalation and exploitation of non-trivial service configurations, clearly benefit from the enhanced contextual reasoning introduced via fine-tuning. This observation underscores the critical role of model specialization in addressing the inherent unpredictability and diversity of real-world penetration testing environments. In summary, the No-RAG evaluation confirms that **xOffense-Qwen3-32B-finetune** can autonomously achieve competitive, and in some cases state-of-the-art, performance in realistic offensive security scenarios, even without external retrieval augmentation. This establishes a robust performance baseline for subsequent RAG-enhanced evaluations.

6.3.2. Performance with Retrieval-Augmented Generation (RAG)

When augmenting the evaluation with the Knowledge Repository module, a substantial shift in performance trends emerges across the six real-world exploitation targets (see Fig. 4). Compared to the baseline (without RAG), the Qwen3-32B-Finetune model demonstrates marked improvement, achieving perfect completion scores on Victim1 and WestWild (1.00) and notable gains on Library2 (+0.20) and Symfonos2 (+0.16). Similarly, moderate increases are observed for Sar and Funbox, reflecting the model's enhanced capability to navigate multi-step attack chains when supported by targeted, contextually relevant prior knowledge.

The gains are less pronounced for Qwen3-32B-Base, with performance remaining comparatively low on challenging targets such as Symfonos2 (0.13) and WestWild (0.25). This disparity underscores the role of fine-tuning in maximizing the benefits of retrieval augmentation—without alignment to domain-specific exploitation strategies, the retrieved information alone is insufficient to ensure consistent execution success. When compared against VulnBot baselines, Qwen3-32B-Finetune with RAG achieves competitive or superior results in four out of six targets, matching the best baseline performance on Library2 (0.80) and surpassing it on Victim1 and WestWild. This suggests that RAG integration not only mitigates the limitations of the base model but also allows the fine-tuned variant

to close the gap—or in certain scenarios, outperform—human-assisted frameworks.

These improvements can be attributed to three key factors: (1) the retriever's ability to surface high-relevance exploitation procedures from a curated cybersecurity corpus, (2) the fine-tuned model's capacity to integrate external information into coherent multi-step reasoning, and (3) the reduction of hallucination-driven dead ends, which are particularly detrimental in constrained exploitation environments. Collectively, these findings reinforce the notion that RAG is a critical enabler for scalable, high-fidelity automated penetration testing in complex real-world settings.

7. Threats to Validity

7.1. Internal validity

The fine-tuning of Qwen3-32B on a Chain-of-Thought—enriched penetration testing dataset introduces potential internal threats. Certain vulnerability classes and exploitation strategies are disproportionately represented, which may bias the model toward specific attack vectors while limiting its capacity to generalize to underrepresented scenarios. Moreover, the integration of prompting strategies and toolchains may embed implicit task-specific heuristics, raising the possibility that reported improvements partly reflect dataset artifacts rather than genuine reasoning ability. Such factors must be considered when interpreting performance gains on structured benchmarks.

7.2. External validity

The evaluation settings—AutoPenBench and AI-Pentest-Benchmark—approximate realistic penetration workflows yet cannot fully capture the heterogeneity of production-scale environments. Operational networks often exhibit greater variability in topology, non-standard configurations, active defenses, and deception mechanisms that remain absent from current benchmarks. In addition, adversarial tactics evolve over time, whereas benchmarks are necessarily static. Consequently, the generalizability of results to enterprise systems, heterogeneous infrastructures, or zero-day exploitation scenarios should be regarded with caution.

7.3. Construct validity

Task completion rate and exploitation success were employed as primary evaluation metrics. While suitable for quantifying functional effectiveness, these measures neglect other dimensions that are central to penetration testing practice. Attributes such as stealth, efficiency of resource utilization, time-to-compromise, and resilience against detection are critical to operational realism yet remain unaccounted for in the adopted benchmarks. Furthermore, binary success measures fail to capture partial progress or incremental compromise, potentially obscuring nuances in agent behavior across complex exploitation chains.

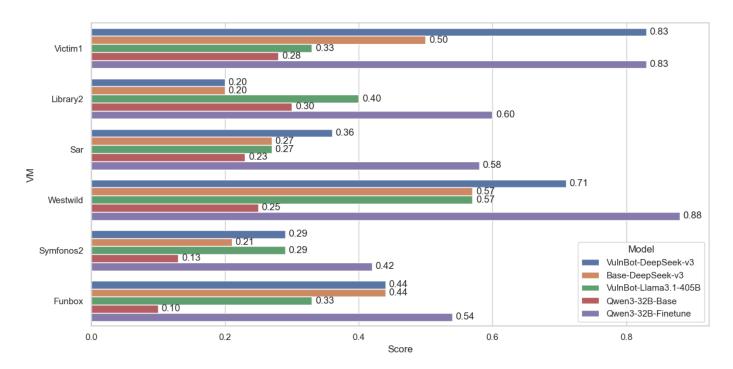


Figure 3: Comparison of subtask completion rates across six real-world vulnerable machines in a No-RAG setting.

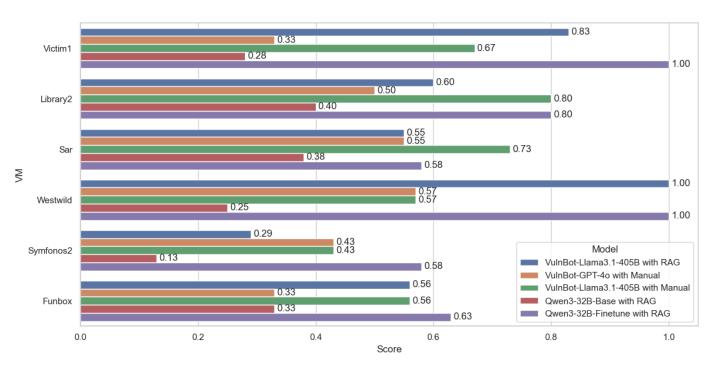


Figure 4: Comparison of subtask completion rates across six real-world vulnerable machines with RAG setting.

7.4. Reliability

The reproducibility of results may be affected by stochastic factors inherent in both large language model inference and auxiliary system tools. Hardware variation, runtime conditions, network latency, and nondeterministic outputs from scanning utilities can yield divergent agent behaviors even under identical inputs. Standardized configurations and repeated trials mitigate these effects but do not eliminate them entirely, implying

that replications across platforms or over extended periods may observe non-negligible variance.

7.5. Summary

In sum, although the reported findings provide strong evidence of the capabilities of xOffense, these validity concerns underscore the need for broader empirical validation. Expanding evaluations to encompass more diverse infrastructures, ad-

versarially adaptive defenses, and richer performance metrics would strengthen the robustness, scalability, and practical applicability of autonomous penetration testing systems.

8. Conclusion and Future Work

This work presented xOffense, an independent, fully autonomous multi-agent framework for penetration testing, designed to address persistent limitations in existing systems such as context loss, limited reasoning continuity, and dependence on large proprietary models. By integrating a fine-tuned, midscale open-source LLM (Qwen3-32B) with a novel grey-box phase prompting mechanism and a purpose-built orchestration architecture, xOffense achieves accurate multi-stage decision-making and robust tool integration across the entire penetration testing lifecycle.

Our evaluation on *AutoPenBench* and *AI-Pentest-Benchmark* demonstrated that xOffense consistently outperforms both larger commercial LLMs (e.g., GPT-40, LLaMA3-70B) and leading open-source baselines (e.g., PentestGPT, VulnBot-LLaMA3-405B). The framework attained an overall task completion rate of 72.72% and a sub-task completion rate of up to 79.17%, while successfully exploiting complex real-world targets. These results highlight that a domain-specialized, mid-scale model—when paired with targeted reasoning guidance—can match or exceed the capabilities of state-of-the-art large-scale systems, offering a cost-effective and reproducible solution for autonomous offensive security operations.

Future work will explore three main directions. First, we aim to optimize the command generation module, potentially through structured function calling, to further improve execution precision. Second, we plan to enhance the robustness of long-running process handling and strengthen the retrieval-augmented generation mechanism with automated updates from vulnerability intelligence sources such as ExploitDB and GitIngest. Third, we intend to extend xOffense's capabilities to support advanced web and GUI interactions via browser automation, enabling it to tackle a broader range of penetration testing scenarios.

Acknowledgement

This research is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number NCM2025-26-01.

Acknowledgement

This research was supported by The VNUHCM-University of Information Technology's Scientific Research Support Fund.

References

[1] Infosecurity Magazine. Nvd revamps operations amid eve surge. https://www.infosecurity-magazine.com/news/ nvd-revamps-operations-eve-surge/, 2024. Accessed: 2025-07-30

- [2] GBHackers. Nist facing challenges in managing cve backlog. https://gbhackers.com/ nist-facing-challenges-in-managing-cve-backlog/, 2024. Accessed: 2025-07-30.
- [3] Isao Takaesu and Daisuke Chikamori. Deep exploit. https: //www.blackhat.com/us-18/arsenal/schedule/index.html# deep-exploit-11908, 2018. Presented at Black Hat USA 2018 Arsenal, Las Vegas. Accessed: 2025-07-30.
- [4] Rapid7. Metasploit penetration testing software, pen testing security. https://www.metasploit.com/, 2024. Accessed: July 27, 2024.
- [5] Abdul Samad, Saad Altaf, and M Junaid Arshad. Advancements in automated penetration testing for iot security by leveraging reinforcement learning. evaluation, 8:9, 2024.
- [6] Khuong Tran, Ashlesha Akella, Maxwell Standen, Junae Kim, David Bowman, Toby Richer, and Chin-Teng Lin. Deep hierarchical reinforcement agents for automated penetration testing. arXiv preprint arXiv:2109.06449, 2021.
- [7] 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, 2024. USENIX Association.
- [8] Xiangmin Shen, Lingzhi Wang, Zhenyuan Li, Yan Chen, Wencheng Zhao, Dawei Sun, Jiashui Wang, and Wei Ruan. Pentestagent: Incorporating Ilm agents to automated penetration testing. In *Proceedings of the* 20th ACM Asia Conference on Computer and Communications Security, pages 375–391, 2025.
- [9] He Kong, Die Hu, Jingguo Ge, Liangxiong Li, Tong Li, and Bingzhen Wu. VulnBot: Autonomous penetration testing for a multi-agent collaborative framework. arXiv preprint arXiv:2501.13411, Jan 2025.
- [10] Luca Gioacchini, Marco Mellia, Idilio Drago, Alexander Delsanto, Giuseppe Siracusano, and Roberto Bifulco. Autopenbench: Benchmarking generative agents for penetration testing, 2024.
- [11] Isamu Isozaki. Ai-pentest-benchmark: A benchmark for automated penetration testing. https://github.com/isamu-isozaki/AI-Pentest-Benchmark, 2024. GitHub repository. Accessed: 2025-07-30.
- [12] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025.
- [13] Gordon Lyon. Nmap: The network mapper free security scanner. https://nmap.org/, 2024. Accessed: July 27, 2024.
- [14] Chris Sullo. Nikto web server scanner. https://github.com/sullo/ nikto, 2024. Accessed: July 27, 2024.
- [15] WPScan Team. Wpscan wordpress security scanner. https://github. com/wpscanteam/wpscan, 2024. Accessed: July 27, 2024.
- [16] Ryusei Maeda and Mamoru Mimura. Automating post-exploitation with deep reinforcement learning. Computers & Security, 100:102108, 2021.
- [17] Van-Hau Pham, Hien Do Hoang, Phan Thanh Trung, Van Dinh Quoc, Trong-Nghia To, and Phan The Duy. Raiju: Reinforcement learningguided post-exploitation for automating security assessment of network systems. *Computer Networks*, 253:110706, 2024.
- [18] 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 cyberattacks. arXiv preprint arXiv:2403.01038, 2024.
- [19] Hanzheng Dai, Yuanliang Li, Zhibo Zhang, and Jun Yan. Refpentester: A knowledge-informed self-reflective penetration testing framework based on large language models. arXiv preprint arXiv:2505.07089, 2025.
- [20] Sho Nakatani. Rapidpen: Fully automated ip-to-shell penetration testing with llm-based agents. arXiv preprint arXiv:2502.16730, 2025.
- [21] Dominik M. Weber, Ioannis Tzachristas, and Aifen Sui. Perses: Unlocking privilege escalation for small llms via extensible heterogeneity. In Proceedings of the 20th ACM Asia Conference on Computer and Communications Security (ASIA CCS '25). ACM, 2025.
- [22] Richard Fang, Rohan Bindu, Akul Gupta, and Daniel Kang. Llm agents can autonomously exploit one-day vulnerabilities. *arXiv preprint* arXiv:2404.08144, 2024.
- [23] Yuxuan Zhu, Antony Kellermann, Akul Gupta, Philip Li, Richard Fang, Rohan Bindu, and Daniel Kang. Teams of Ilm agents can exploit zero-day vulnerabilities. arXiv preprint arXiv:2406.01637, Mar 2025.

- [24] Lajos Muzsai, David Imolai, and András Lukács. Hacksynth: Llm agent and evaluation framework for autonomous penetration testing. arXiv preprint arXiv:2412.01778, 2024.
- [25] Minghao Shao, Sofija Jancheska, Meet Udeshi, Brendan Dolan-Gavitt, Haoran Xi, Kimberly Milner, Boyuan Chen, Max Yin, Siddharth Garg, Ramesh Karri, Prashanth Krishnamurthy, Farshad Khorrami, and Muhammad Shafique. Nyu ctf bench: A scalable open-source benchmark dataset for evaluating Ilms in offensive security. In NeurIPS 2024 Datasets and Benchmarks Track, 2024.
- [26] Yuxuan Zhu, Antony Kellermann, Dylan Bowman, Philip Li, Akul Gupta, Adarsh Danda, Richard Fang, Conner Jensen, Eric Ihli, Jason Benn, Jet Geronimo, Avi Dhir, Sudhit Rao, Kaicheng Yu, Twm Stone, and Daniel Kang. Cve-bench: A benchmark for ai agents' ability to exploit realworld web application vulnerabilities. arXiv preprint arXiv:2503.17332, Mar 2025
- [27] Isamu Isozaki, Manil Shrestha, Rick Console, and Edward Kim. Towards automated penetration testing: Introducing LLM benchmark, analysis, and improvements. In *Proceedings of the 2025 ACM Conference* (companion/adjunct) on Computer and Communications Security, 2025. Accessed: 2025-08-06.
- [28] Julius Henke. Autopentest: Enhancing vulnerability management with autonomous Ilm agents. arXiv preprint arXiv:2505.10321, 2025.
- [29] Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for "mind" exploration of large language model society. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, Advances in Neural Information Processing Systems, volume 36, pages 51991–52008. Curran Associates, Inc., 2023.
- [30] Liu Qian, Song Jinke, Huang Zhiguo, Zhang Yuxuan, glide the, and liunux4odoo. langchain-chatchat. GitHub repository, 2024.
- [31] DirB Project. Dirb web content scanner. https://gitlab.com/ kalilinux/packages/dirb, 2025.
- [32] Gobuster Project. Gobuster directory/file, dns and vhost busting tool written in go. https://github.com/OJ/gobuster, 2025.
- [33] OWASP Amass Project. Owasp amass in-depth attack surface mapping and asset discovery. https://github.com/owasp-amass/amass, 2025.
- [34] sqlmap Developers. sqlmap automatic sql injection and database takeover tool. https://github.com/sqlmapproject/sqlmap, 2025.
- [35] THC Hydra Team. The-hydra network logon cracker. https://github.com/vanhauser-thc/thc-hydra, 2025.
- [36] Openwall Project. John the ripper password cracker. https://github.com/openwall/john, 2025.
- [37] Offensive Security. Exploit database (exploit-db). https://www.exploit-db.com/, 2025.
- [38] Carlos Polop. Hacktricks: Hacking techniques & privilege escalation encyclopedia. https://book.hacktricks.xyz/, 2025.
- [39] Raj Chandel and Hacking Articles Team. Hacking articles: A cyber security community blog. https://www.hackingarticles.in/, 2025.
- [40] Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. arXiv preprint arXiv:2507.02259, 2025.
- [41] Offensive Security. Kali linux: Penetration testing and ethical hacking linux distribution. https://www.kali.org/, 2025.
- [42] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–16. IEEE, 2020.
- [43] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. Advances in neural information processing systems, 35:16344–16359, 2022.
- [44] TryHackMe Team. Tryhackme: Hands-on cybersecurity training platform. https://tryhackme.com, 2024.
- [45] HackTheBox Team. Hack the box: Cybersecurity labs and challenges. https://www.hackthebox.com, 2024.
- [46] VulnHub Community. Vulnhub: Vulnerable machines for penetration testing practice. https://www.vulnhub.com, 2024.
- [47] HuggingFace Team. Huggingface datasets hub: Open-source datasets for machine learning. https://huggingface.co/datasets, 2024.

[48] Migel Tissera and WhiteRabbitNeo Team. Whiterabbitneo cybersecurity dataset (wrn-chapter-1, wrn-chapter-2). https://huggingface.co/datasets/WhiteRabbitNeo/WRN-Chapter-1, 2024.