Information Fusion 127 (2026) 103731

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

# PTFusion: LLM-driven context-aware knowledge fusion for web penetration testing

Wenhao Wang [a], Hao Gu [b], Zhixuan Wu [b], Hao Chen [b], Xingguo Chen [b], Fan Shi [a,*]

[a] College of Electronic Engineering, National University of Defense Technology, Hefei, 230037, Anhui, China
[b] Nanjing University of Posts and Telecommunications, Nanjing, 210009, Jiangsu, China

ARTICLE INFO

*Keywords:*
Large language models
Web penetration testing
Multi-agent systems
Dynamic knowledge graphs
Chain-of-Thought

ABSTRACT

This paper presents PTFusion, an LLM-driven web penetration testing framework that addresses inefficient task guidance and imprecise command execution challenges in web penetration testing. Employing a semi-decentralized multi-agent collaborative architecture, PTFusion maintains strategic coherence while enabling autonomous tactical execution, and uses the Model Context Protocol to more conveniently call different types of penetration testing tools. To effectively guide task execution, the PTFusion designs a context-aware knowledge fusion mechanism to plan tasks based on the dynamic knowledge graph and executed actions, and uses the preference-based chain-of-thought prompting to address the issue of redundant and difficult to align outputs from different types of penetration testing tools. Compared to methods like PentestGPT, PTFusion demonstrates significantl superior performance in both task completion effectiveness and stability. The context-aware knowledge fusion mechanism enables PTFusion to conduct more precise strategic planning and execute penetration testing commands with greater accuracy, ensuring reliable completion of web penetration testing tasks across various scenarios.

## 1. Introduction

The rapidly evolving sophistication of cyber threats requires defensive capabilities that can dynamically adapt to novel attack vectors. Modern web applications present increasingly complex attack surfaces characterized by heterogeneous architectures that span various content management systems, development languages, databases, and even virtual environments. In this landscape, manual penetration testing faces fundamental scalability limitations, driving critical demand for intelligent automation solutions capable of conducting comprehensive security assessments. Automated penetration testing has thus emerged as an essential paradigm for proactive vulnerability discovery, promising continuous assessment capabilities that transcend human speed and fatigue limitations. However, existing approaches fundamentally struggle to navigate the dynamic complexity and uncertainty of target web environments, where fragmented reconnaissance data - generated by diverse penetration testing tools - must continuously guide adaptive multistage exploitation.

Reinforcement learning (RL) [1–4] initially presented a theoretically compelling approach for automated penetration testing through adaptive sequential decision-making, and employs methods such as hierarchical reinforcement learning [1,5,6], state and action space dimensionality reduction [2,7–9], and the integration of expert knowledge [3,10] to enhance the performance of the generated policies. These methods can effectively improve the efficiency of penetration testing in unfamiliar, dynamic, and partially observable environments. However, practical implementations reveal fundamental misalignment with operational realities: RL agents typically rely on static simulated training environments that struggle to model the non-stationary characteristics of target networks, often leading to catastrophic performance degradation and operational forgetting during real-world deployment. More critically, the paradigm's inherent emphasis on policy optimization overlooks penetration testing's core dependency on precise orchestration of specialized security tools (e.g., Nmap, Metasploit, Burp Suite) and contextual fusion of their semantically heterogeneous outputs, where conventional RL architectures exhibit severe deficiencies.

Recent advances in large language models (LLMs) [11,12], demonstrated by models like GPT-4.1-mini [13], GPT-4o-mini [14], and Qwen-72B [15], have shown remarkable success across various domains including code generation, decision automation, and multi-modal reasoning. Building on these developments, LLMs [16] introduce an innovative methodology for penetration testing, enabling dynamic orchestration of security tools and multi-modal analysis of their outputs. Recent systems like PentestGPT [17] and PentestAgent [18] demonstrate

* Corresponding author.
*E-mail address:* shifan17@nudt.edu.cn (F. Shi).

https://doi.org/10.1016/j.inffus.2025.103731
Received 30 June 2025; Received in revised form 13 August 2025; Accepted 9 September 2025
Available online 13 September 2025

preliminary task comprehension capabilities; however, they remain confined to human-in-the-loop paradigms and simplistic penetration scenarios without fully demonstrating their operational potential in real-world testing environments.

However, in the process of penetration testing, modern security tools (Nmap, Metasploit, Burp Suite) generate heterogeneous data streams that include network topologies, service banners, vulnerability signatures, and exploit metadata. This syntactic and semantic fragmentation causes three operational failures: (1) *imprecise command execution* where LLMs struggle to generate precise executable commands directly from contextual information, (2) *context decay* where critical relationships between ports, services, and vulnerabilities are lost during handovers, and (3) *inefficient task guidance* where analysts may be trapped in feedback loops of localized analysis without unified correlation frameworks. These limitations position LLM-enhanced penetration testing as both a groundbreaking opportunity and formidable research frontier, particularly with the emergence of the Model Context Protocol (MCP), which enables LLMs to seamlessly orchestrate external tools and process heterogeneous information into actionable penetration testing knowledge.

To overcome these limitations, this paper proposes *PTFusion* - a semi-decentralized multi-agent coordination system to improve penetration testing with the LLM. PTFusion leverages MCP to dynamically invoke external tools and process diverse information sources, ensuring real-time knowledge updates and facilitating unified correlation of findings. The contributions of this work are as follows.

(1) **The MCP-enabled Semi-Decentralized Multi-Agent Coordination Framework**. The proposed framework employs a hierarchical multi-agent architecture, where the *MasterAgent* is responsible for strategic planning, and the *ReconAgent* and *AttackAgent* have the tactical execution autonomy, enabling it to autonomously determine whether to continue a task based on contextual information. All three agents can invoke distinct tools through their respective MCP Servers to collaboratively execute penetration testing tasks.
(2) **The Context-Aware Knowledge Fusion Mechanism**. This mechanism employs a context-aware dynamic knowledge graph that dynamically models entities (hosts, ports, vulnerabilities) and their relationships to guide task planning, and uses the preference-based Chain-of-Thought Prompting to address noisy tool outputs, ensuring all intelligence fed into the system maintains verifiable ground-truth integrity for reliable autonomous decision-making.
(3) **Empirical Validations**. The results indicate that, unlike PentestGPT whose performance is closely tied to the operator's experience, PTFusion autonomously completes penetration testing tasks across all tested environments with consistent stability.

The remainder of this paper is structured as follows: Section 2 introduces the background of this paper. Section 3 details our semi-decentralized multi-agent framework including agent roles and MCP-based tool calling. Section 4 introduces our context-aware knowledge fusion mechanism with dynamic task planning and schema-based data alignment. Section 5 presents experimental validation. Finally, the last section concludes the work.

## 2. Background

### 2.1. Model context sprotocol

In penetration testing, analysts rely on various cybersecurity tools, each with distinct objectives and complex, environment-dependent configurations. Programmatic invocation is challenging due to varied command parameters, execution contexts, and heterogeneous output formats. While these challenges are common across many domains, the Model Context Protocol (MCP) offers a standardized interface for tool invocation, output parsing, and context management, significantly reducing integration complexity.

Originally proposed by Anthropic in November 2024 [19] as a general, model-agnostic specification to enable seamless integration of LLMs with external tools, datasets, and structured contexts, MCP defines a unified communication layer supporting stateful interactions among three roles: the MCP Host, the MCP Client, and the MCP server. Although not developed specifically for penetration testing, MCP's flexible architecture and abstractions render it well-suited to the dynamic, heterogeneous tool environments characteristic of cybersecurity workflows.

The MCP Host orchestrates workflow execution and mediates interactions between the LLM, Client, and Server. The MCP Server manages access to structured data resources and executes actions via Tools, acting as a bridge to external ecosystems. Communications are performed through JSON-RPC requests, responses, and notifications.

The cybersecurity domain features diverse tools, proprietary interfaces, and heterogeneous data formats (e.g., Nmap XML, Nessus scans, Metasploit outputs). MCP mitigates this heterogeneity by exposing disparate tools as standardized MCP Tools and converting varied security data into structured MCP Resources. For example, an MCP Tool for Nmap encapsulates scan parameters in a JSON schema, while an MCP Resource normalizes outputs into a consistent structure, enabling LLMs to consume data without tool-specific parsing. Similarly, Metasploit's MCP Tool abstracts exploit configurations through a uniform interface. This abstraction allows LLMs to discover, securely access, and interact with diverse security tools through a single protocol, irrespective of their architecture.

MCP's modularity, extensibility, and standardized interaction mechanisms offer distinct advantages for integrating LLMs with dynamic, tool-intensive workflows. Automated penetration testing, which requires rapid tool coordination, context-aware decisions, and adaptive execution, magnifies these benefits. Applying MCP in this domain extends its role from generic tool integration to supporting time-sensitive security operations with strict environment-specific configurations. This enables seamless tool interoperability without manual reformatting, establishing MCP as the backbone of an intelligent, adaptive penetration testing framework.

### 2.2. Dynamic Knowledge Graph

A Dynamic Knowledge Graph (DKG) is a temporally evolving knowledge representation that supports real-time aggregation, adaptive inference, and semantic enrichment by capturing both current and historical entity states. Unlike static graphs, DKGs continuously update time-dependent relationships, enabling reasoning over rapidly changing contexts-an essential capability in dynamic domains.

Prior studies have explored temporal reasoning in evolving graphs, including RL-based traversal optimization, temporal community detection, and large-scale pipelines that integrate structured and conversational data [20–22]. In addition, LLM-graph integration methods such as Debate on Graphs [23] demonstrate that subgraph focusing and iterative refinement can suppress false positives and improve inference fidelity, principles equally valuable in operational settings.

In penetration testing, situational awareness evolves continuously as reconnaissance, scanning, and exploitation actions reveal new hosts, services, or vulnerabilities. DKGs naturally accommodate such incremental updates while preserving temporal coherence, enabling agents to correlate past and present findings, anticipate changes, and adjust tactics dynamically. When combined with LLM-based entity extraction, debate-inspired refinement, and predictive temporal signals, they transform heterogeneous, noisy tool outputs into a consistent, queryable, and temporally coherent knowledge base. This robust substrate underpins the context-aware knowledge fusion mechanism in PTFusion, ensuring

that strategic planning is informed by accurate, up-to-date, and context-rich intelligence.

### 2.3. Chain-of-Thought Prompting

In addressing challenges of large language model (LLM) deployment, existing literature has explored complementary approaches such as federated transfer learning (for on-device specialization) and privacy-preserving aggregation (for edge environments). These methods primarily focus on model adaptation and data protection at the training or deployment level, requiring adjustments to model weights, retraining, or modifications to deployment topology [16,24].

Against this backdrop, Chain-of-Thought Prompting (CoT) emerges as a distinct prompting strategy: it elicits explicit, stepwise reasoning traces from LLMs by requesting intermediate reasoning steps prior to a final answer. Critically, CoT operates at inference time and is model-agnostic-structuring the model's internal computation without altering weights, retraining, or deployment architecture. This scope difference enables prompt-level scaffolding that guides reasoning trajectories on a per-query basis, supporting rapid, context-sensitive input transformation.

In penetration testing, where raw tool outputs are often voluminous, heterogeneous, or cluttered with irrelevant/ambiguous information, CoT mitigates these issues by directing models to perform explicit intermediate steps (e.g., entity extraction, normalization, prioritized summarization). This aligns with findings from multi-modal reasoning research, which emphasize constrained output formats and evaluation-aware prompt design as key to avoiding spurious correlations and failure modes [25,26].

Preference-based Chain-of-Thought Prompting further refines this paradigm by encoding operator preferences, policies, and task-specific heuristics into prompts. When integrated with the DKG and MCP, it functions as a lightweight, interpretable adapter.

When integrated with Dynamic Knowledge Graph (DKG) updates and Model Context Protocol (MCP)-managed tool invocations, it functions as a lightweight, interpretable adapter: generating verifiable, structured results that can be directly ingested by DKG rules or MCP tools, thereby reducing the processing burden of varied penetration testing tool outputs.

## 3. LLM-driven automated penetration testing framework

In this section, we introduce our LLM-driven automated penetration testing framework, detailing its collaborative task execution architecture and tool calling methodology.

### 3.1. The semi-decentralized multi-agent collaborative architecture

Our system employs a hierarchical, MCP-enabled semi-decentralized multi-agent collaborative architecture. The framework is designed to integrate the guidance of global strategic planning with the flexibility of local tactical execution. The core of this architecture consists of three types of functionally distinct agents: a *MasterAgent*, a *ReconAgent*, and an *ReconAgent*. The specific collaborative process is shown in Fig. 1. Prefixes M-, R-, and A- in the Fig. 1 indicate different steps in the workflow of the *MasterAgent*, *ReconAgent*, and *AttackAgent*, and the dashed lines represent the communication between agents.

#### 3.1.1. The MasterAgent's workflow

The *MasterAgent* serves as the strategic decision-making node, responsible for analyzing user intent, formulating high-level penetration testing objectives and dynamically adjusting them. It does not directly intervene in specific tool selection or command execution; instead, it guides the penetration testing process by assigning high-level tasks to subordinate agents, such as "exploring potential vulnerabilities in a specific web application." Its workflow can be divided into five steps.

(1) **M-1 Acquire Knowledge.** Retrieve updated information from the Dynamic Knowledge Graph and MCP Server to build situational awareness of the target environment.
(2) **M-2 Return Queries.** Return queries to the *MasterAgent*.
(3) **M-3 Provide Knowledge and Action History.** Provide relevant knowledge and previous action records to support task planning.
(4) **M-4 Return Task Planning.** Generate a concrete, high-level strategic step based on the reasoning capabilities of large language models. This part of the work will be elaborated in Section 4.1.
(5) **M-5 Allocate Recon (or Attack) Tasks.** Allocate specific reconnaissance or attack tasks to corresponding agents.

#### 3.1.2. The ReconAgent and AttackAgent's workflow

The *ReconAgent* and *AttackAgent*, acting as tactical execution agents, are endowed with a high degree of autonomy. Upon receiving a macroscopic task from the MasterAgent, each subordinate agent activates its tactical execution engine, operating autonomously within an independent, short-term action history context. Both the *ReconAgent* and *AttackAgent* follow the same workflow. We use the *ReconAgent* as an example to illustrate its operational steps.

(1) **R-1 Analyze the Task.** Interpret the assigned task and analyze its scope.
(2) **R-2 Request Tactical Decisions.** The *ReconAgent* first utilizes the LLM to determine whether its assigned task should proceed. If continuation is required, it submits tactical requests to the LLM for operational guidance.
(3) **R-3 Return Execution Commands.** Generate concrete commands to run specific tools.
(4) **R-4 Invoke MCP Server.** Call the tools in MCP Server based on recommendations from the LLM.
(5) **R-5 Use Tools to Execute Commands.** Use tools to execute payloads for specific targets.
(6) **R-6 Return Recon (or Attack) Results.** The *ReconAgent* receives the execution results from the tools.
(7) **R-7 Submit Final Summary.** After the *ReconAgent* determines that the task has been fully executed, it will provide a final summary of the process executed in this episode to the *MasterAgent* agent.

The core capability of the *ReconAgent* and *AttackAgent* lies in their autonomous task execution authority. *The ReconAgent*, for example, is primarily responsible for mapping the target's digital attack surface. When tasked by the *MasterAgent* to "conduct reconnaissance on a web application", it does not passively await specific instructions but initiates its internal tactical loop. In the first round, lacking local context, it leverages the LLM to prioritize foundational tools like Nmap for a port scan. Once the result-such as an open port 80-is logged in its temporary task history, this new context is fed back to the LLM in the next cycle. Based on this key intelligence, the LLM then autonomously decides to execute more targeted actions, like a dirb directory enumeration or a web vulnerability scan, all without any intervention from the *MasterAgent*.

#### 3.1.3. Motivations for designing semi-decentralized multi-agent architecture

The above closed-loop process of "Strategic Guidance ↓ Tactical Autonomy ↓ Outcome Reporting ↓ Strategic Adjustment" ensures that the *MasterAgent*'s decisions are always based on the most authoritative and comprehensive battlefield information, while also granting subordinate agents sufficient flexibility and efficiency at the tactical level. This allows the system to maintain a clear strategic direction while demonstrating a high degree of adaptability and robustness in dynamic penetration testing scenarios, primarily due to two rationales.

*Rationale for Semi-Decentralized Architecture Design.* The adoption of a semi-decentralized architecture stems from the need to address fundamental challenges in complex penetration testing scenarios. A purely centralized model, while offering strong global coordination, risks becoming a bottleneck, hindering responsiveness to dynamic findings

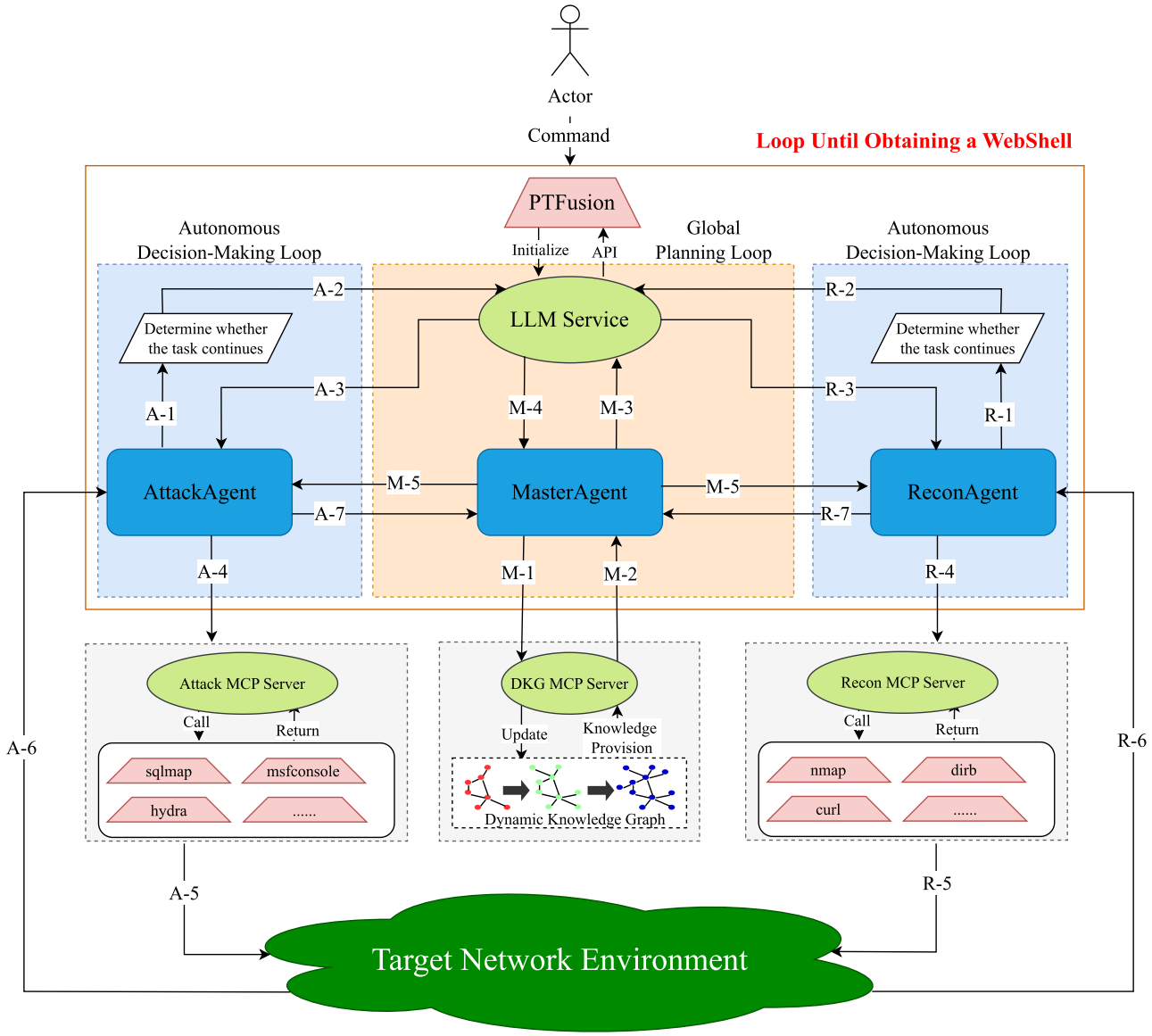


**Fig. 1.** The MCP-enabled semi-decentralized multi-agent collaborative penetration testing framework.

and increasing vulnerability to single points of failure. Conversely, a fully decentralized system, despite its robustness and adaptability, may suffer from strategic incoherence, task duplication, or conflicting actions among agents lacking a unified objective. Our hierarchical, semi-decentralized approach represents an optimized trade-off. It leverages the *MasterAgent* to provide essential top-down strategic oversight, goal formulation, and dynamic adaptation based on the mission context and aggregated results. More importantly, it empowers the subordinate *ReconAgent* and *AttackAgent* with significant autonomy simultaneously. This autonomy enables them to efficiently execute specialized tactical operations, react in real-time to emergent local conditions (e.g., unexpected reconnaissance findings), and make localized decisions regarding tool selection and command execution without requiring constant micro-management from the center. This structure effectively balances the requirement for strategic coherence with the necessity for agile,localized tactical execution in unpredictable environments.

*Rationale for the Tripartite Agent Architecture.* The decision to employ precisely three distinct agent types is grounded in the core functional decomposition of the penetration testing workflow and the principle of separation of concerns. Two agents would necessitate an undesirable functional conflation: merging the strategic planning and oversight role (*MasterAgent*) with either reconnaissance or attack functions would overload a single agent, blurring responsibilities and hindering scalability. Combining the reconnaissance (*ReconAgent*) and attack (*AttackAgent*) roles within a single agent would undermine modularity and violate the critical logical separation between the information gathering/assessment phase and the active exploitation phase, potentially leading to premature or contextually inappropriate actions. Introducing a fourth specialized agent (e.g., dedicated reporting, vulnerability analysis, or persistence) was evaluated but deemed to introduce unnecessary complexity at the foundational architectural level for our initial objectives. The *MasterAgent* naturally incorporates the role of synthesizing results and high-level reporting based on inputs from the tactical agents.The *ReconAgent* is singularly focused on discovery, enumeration, and vulnerability identification, providing the essential intelligence foundation. The *AttackAgent* is dedicated to the selective and controlled exploitation of identified vulnerabilities to achieve specific objectives. In subsequent penetration phases such as lateral movement and privilege escalation, the collaborative approach among multiple agents may differ. This scenario might necessitate introducing additional

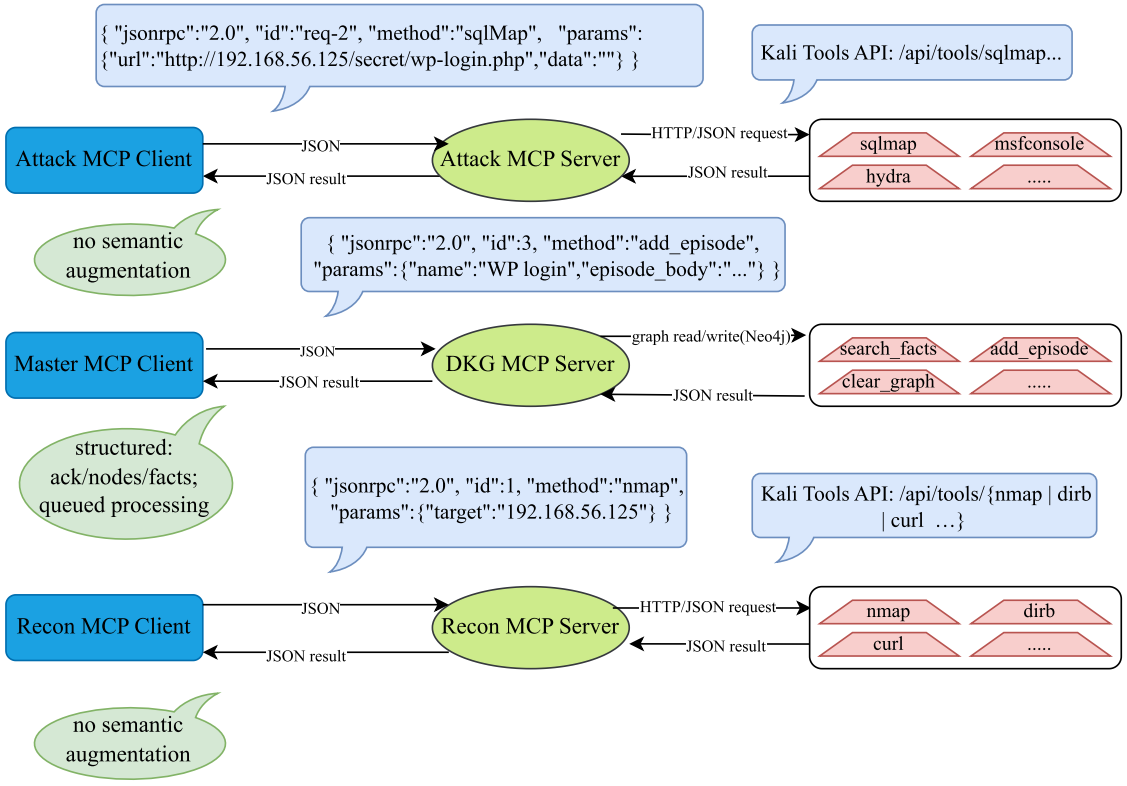


**Fig. 2.** The interactions between different MCP clients, servers and tools.

penetration testing agents like Command & Control Agents or Privilege Escalation Agents.

### 3.2. MCP-Enabled tool calling

In the proposed framework, we use the Model Context Protocol (MCP) to call different types of penetration testing tools. By rigorously decoupling cognitive tasks from kinetic actions and by establishing specialized servers for distinct operational domains, the framework establishes a modular and extensible architecture for automated penetration testing. Each agent calls a server based on its specific role, as detailed in Table 1.

Where, the *MasterAgent* exclusively calls the DKG MCP Server to query and store information, using the knowledge graph and historical actions to support its task planning. The *ReconAgent* calls the Recon MCP Server to execute information gathering tools like `nmap` and `dirb` in order to identify and map the target's attack surface. The *AttackAgent* calls the Attack MCP Server to execute exploitation tools like `msfconsole` and `sqlmap`.

Fig. 2 shows the interactions between different MCP clients, servers and tools. The agent, acting as an MCP client, issues a JSON request consisting of a named tool and explicit parameters. The corresponding MCP server validates the method and parameters, executes the invocation, and returns a structured JSON response. To keep downstream steps directly usable, the Recon/Attack servers forward the JSON returned by the controlled execution backend without semantic augmentation. The DKG server enqueues write requests and immediately returns an acknowledgment (e.g., the current queue position), while entity/relationship extraction and persistence are performed asynchronously; on retrieval, it returns structured JSON at the node and fact levels, without introducing tool-external inferences.

*Recon MCP Server (information gathering).* The Recon MCP Server exposes reconnaissance actions as uniform tool interfaces, including port/service identification, directory and path enumeration, and HTTP probing. After the client specifies the tool and a minimal parameter set in JSON, the server translates the call into an HTTP/JSON request to a controlled execution backend and, upon receiving the backend's JSON output, returns it to the client as a structured JSON response. The response aligns with the objective outputs of the underlying tools; for example, open ports with service identifiers, directory/path hits with their HTTP status and response size, and salient HTTP response fields (status and key headers). The server does not introduce additional semantics beyond returning the backend output in structured form.

*Attack MCP Server (exploitation).* The Attack MCP Server unifies exploitation and command execution into a non-interactive invocation interface. The client provides the tool (or command) and explicit parameters in JSON; the server translates the call into an HTTP/JSON request to the controlled execution backend and returns the backend's structured JSON output. To support stage progression, the response preserves verifiable indicators such as confirmations of remote file writes or uploads, exit codes, and key evidence contained in standard output. The server does not expand or reinterpret these outputs beyond their structured presentation.

**Table 1**
MCP-enabled tool calling.

| Index | MCP Server | Tools | Description |
|---|---|---|---|
| 1 | DKG | search_facts | Queries the knowledge graph. |
| 2 | | add_episode | Stores structured knowledge into the DKG. |
| 3 | | clear_graph | Clears the knowledge graph. |
| 4 | Recon | Nmap | Scans for open ports and services. |
| 5 | | Dirb | Enumerates web paths and files. |
| 6 | | Curl | Inspects HTTP responses and methods. |
| 7 | Attack | Msfconsole | Executes exploit modules. |
| 8 | | Hydra | Performs brute-force password cracking. |
| 9 | | Sqlmap | Automates SQL injection attacks. |

*DKG MCP Server (knowledge write and retrieval)*. The DKG MCP Server provides a uniform JSON interface for knowledge ingestion and semantic retrieval. Clients submit text, structured JSON, or message-style content; the server enqueues the submission as an event and immediately returns an acknowledgment (e.g., the current queue position), while entity/relationship extraction and persistence are performed asynchronously. On retrieval, the server returns structured JSON results at the node and fact levels that are directly consumable by downstream components. To preserve per-namespace ordering, writes sharing the same `group_id` are processed serially in queue order. Externally, the interface consistently exposes JSON requests and JSON responses.

## 4. Context-aware knowledge fusion mechanism

In the proposed automated penetration testing framework, we employ a context-aware knowledge fusion mechanism to integrate and harmonize the intelligence gathered from various penetration testing tools. This mechanism is designed to ensure that the fused knowledge is both accurate and relevant to the current operational context, thereby enabling more effective guidance for agent task execution.

### 4.1. Task planning with context-aware dynamic knowledge graph

The dynamic knowledge graph is a key component of the proposed framework, which reflects real-time awareness of the target system, thereby supporting analysis of its attack surface. It is constructed and updated in real-time based on the current operational context, and is used to guide the *MasterAgent*'s task planning process, including the target system's configuration, the penetration testing tools used, and the system's current state.

#### 4.1.1. The dynamic knowledge sgraph

The dynamic knowledge graph serves as the core component for the *MasterAgent*'s task planning, utilizing a graph structure to model penetration testing entities-including hosts, ports, services, and vulnerabilities-along with their interconnections. Task execution by either the *ReconAgent* or *AttackAgent* dynamically updates this graph in real time. For example, a successful port scan adds new port nodes and establishes a relationship with the corresponding host, while a successful exploit may create a new credential node associated with a specific service. Fig. 3 shows the dynamic knowledge graph within the penetration testing tasks.

The entities and relationships in the knowledge graph are illustrated in Table 2.

In the dynamic knowledge graph, each entity is assigned some attributes corresponding to the information that the *ReconAgent* and *AttackAgent* may obtain during the execution of tasks. For example, the **Host** entity has attributes such as ip, hostname and operational system, the **Port** entity has attributes such as the port number and current state, and the **Service** entity has attributes such as the service name, version and banner. Each attribute is proposed and aligned through the LLM and schema-based promptings, and the specific work will be introduced in the next section. Table 3 shows the attributes of each entity in the dynamic knowledge graph.

#### 4.1.2. Two-stage task planning

Based on the reasoning and context retrieval capabilities by the dynamic knowledge graph, *The MasterAgent*'s task planning process can be divided into two stages: "Context Retrieval" and "Strategy Generation", as illustrated in Fig. 4.

**Stage One: Context Retrieval - From High-Level Intent to Focused Query**. In each decision cycle, *The MasterAgent* does not blindly retrieve all information; instead, it first utilizes the LLM to determine what question to ask. It provides the LLM with the current high-level objective (e.g., self.current_target) and a summary of past actions (action_history), allowing the LLM to identify the most critical information gap. The LLM's core task is to convert the ambiguous intent of "Which attack surface of this target should I focus on next?" into an executable, focused natural language query. For example, it might generate query phrases like "query port scan and service discovery history for 192.168.1.3" or "search for known vulnerabilities related to the Apache service." This step converges open-ended strategic thinking into a precise data retrieval command.

**Stage Two: Strategy Generation - Logical Reasoning on Precise Context**. After obtaining highly relevant, noise-free contextual information from phase one (i.e., the facts returned by the knowledge graph in response to the query phrase), *The MasterAgent* submits this structured data, along with the high-level objective, to the LLM again. At this point, the LLM's role shifts to that of a strategic planner. Its task is to perform logical reasoning on this clean, focused dataset to generate a concrete, high-level strategic step, such as "Assign task to ReconAgent: perform directory enumeration on port 80 of 192.168.1.3."

**Table 2**
Entities and relationships in the dynamic knowledge graph.

| Index | Subject | Relationships | Object |
|---|---|---|---|
| 1 | Host | HAS_PORT | Port |
| 2 | Port | RUNS | Service |
| 3 | Service | HAS_VULN | Vulnerability |
| 4 | Service | SERVES_WEB | WebSite |
| 5 | WebSite | SERVES_PATH | SensWebPath |
| 6 | SensWebPath | CONTAINS_VULNS | Vulnerability |
| 7 | Host | BELONGS_TO | NetworkSegment |
| 8 | Service | CAN_LOGIN | LoginStatus |
| 9 | WebPath | BRUTEFORCE | BruteforceStatus |

**Table 3**
Attributes of each entity in the dynamic knowledge graph.

| Index | Entity | Attributes | Descriptions |
|---|---|---|---|
| 1 | Host | ip | The IP address. |
| 2 | | hostname | The name of a target host. |
| 3 | | os | Windows, Redhat or Ubuntu. |
| 4 | Port | portnum | 80, 443, 445. |
| 5 | | portocol | TCP/UDP. |
| 6 | | state | Open/Filtered/Closed. |
| 7 | Service | name | The name of the service. |
| 8 | | version | The version of the service. |
| 9 | | banner | The banners of the service. |
| 10 | Vulnerability | cve_id | The cve number. |
| 11 | | vuln_type | Such as RCE/LFI/SQLi/XSS. |
| 12 | WebSite | base_url | The basic URL. |
| 13 | | cms | The content management system. |
| 14 | | auth_types | Such as form_auth/basic_auth/oauth. |
| 15 | | has_login_page | Whether a login page is present. |
| 16 | | has_admin_path | Whether an administrator page is present. |
| 17 | SensWebPath | path | The sensitive url path. |
| 18 | | status_code | The HTTP code. |
| 19 | NetworkSegment | cidr | The IP network segment. |
| 20 | | segment_type | The type of the network segment. |
| 21 | LoginStatus | login_possible | Whether the service can be logged in. |
| 22 | | login_methods | Supported login methods. |
| 23 | | auth_protocol | Such as HTTP/SSH/RDP. |
| 24 | | credentials_found | Whether credentials are found. |
| 25 | BruteforceStatus | bruteforce_possible | Whether the website path is vulnerable. |
| 26 | | captcha | Whether captchas are present. |
| 27 | | rate_limit | The limitations for brute-force attacks. |

### 4.2. Preference-based chain-of-Thought prompting

During the execution of penetration testing tasks, raw outputs from penetration testing tools present a significant obstacle. This data is often verbose, syntactically diverse, and saturated with information that is irrelevant to strategic decision-making. Feeding this heterogeneous, noisy data directly to an LLM induces three critical failures: (1) **Cognitive Overload**, where the LLM's context is saturated with low-value data; (2) **Reasoning Deviation**, where the model fixates on operationally ir-

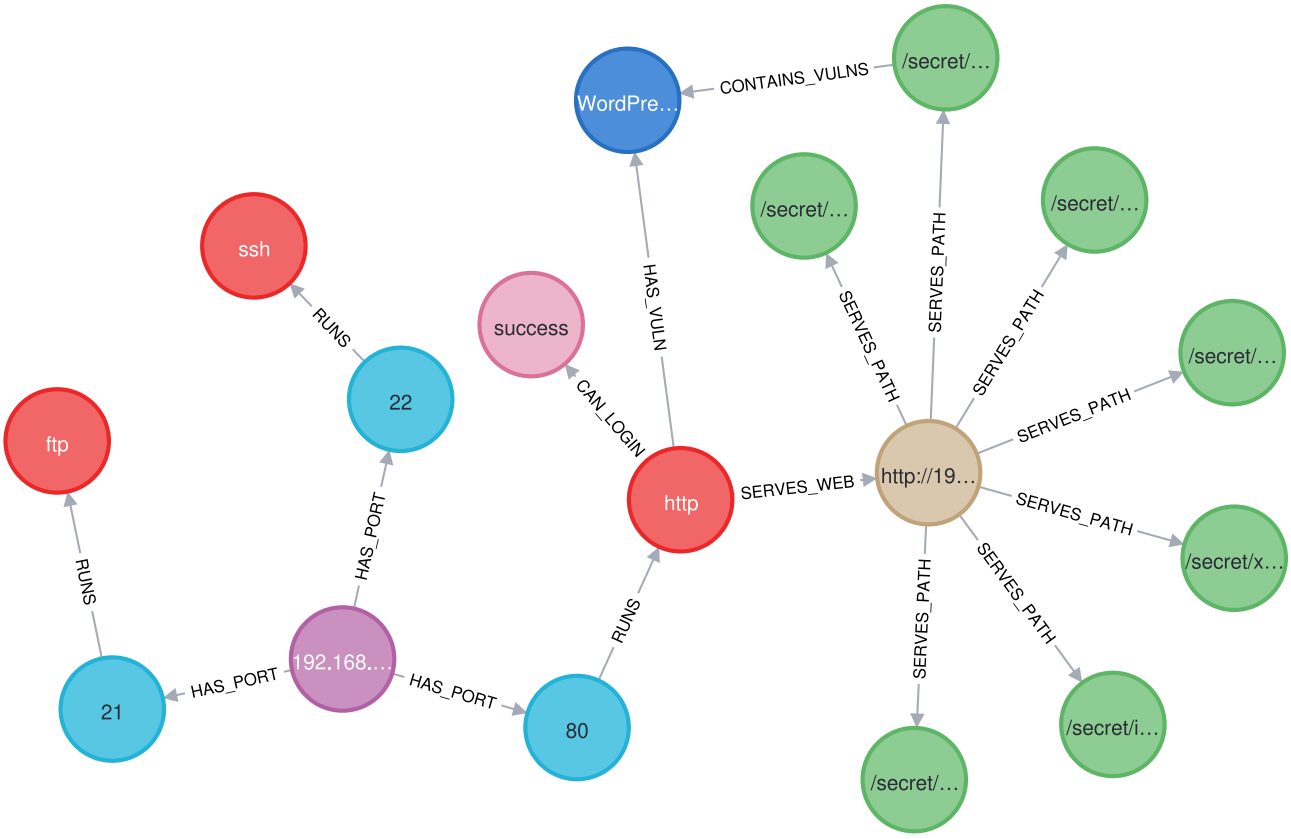


**Fig. 3.** The dynamic knowledge graph within the penetration testing tasks.

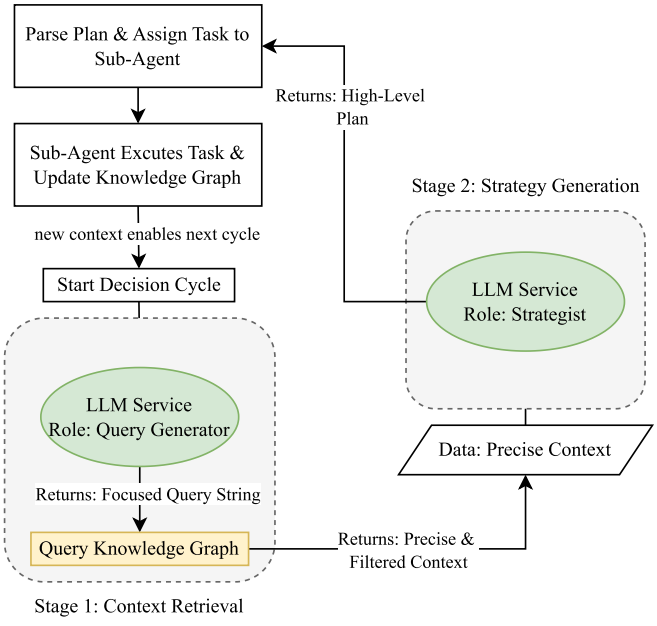


**Fig. 4.** The two-stage decision-making loop of the master agent based on the dynamic knowledge graph.

relevant details; and (3) **Operational Inefficiency**, where the model becomes trapped in analysis loops over noisy data.

To overcome these obstacles, we design and implement a Preference-Based Chain-of-Thought (CoT) Prompting mechanism to address multi-source data fusion and alignment challenges, especially in the `R-6` in the *ReconAgent*'s workflow. We craft unique prompts for each agent role, embedding two core components: 1) Preferences, which set the value orientation and safety boundaries for decisions through explicit rules, and 2) a Chain-of-Thought, which specifies the reasoning process through step-by-step instructions. The detailed implementation is shown in Fig. 5.

To convert raw, unstructured tool outputs into actionable intelligence, our system employs the four-step information alignment process. The initial step, **Strict Deduplication**, consolidates all findings pertaining to the same entity (e.g., file, path, or service) to eliminate data redundancy. Subsequently, **Categorized Aggregation** organizes this deduplicated data into predefined semantic classes, such as "Sensitive Files/Paths", thereby structuring the information for attack surface analysis. The final two stages, **Strict Prohibition of Information Fabrication** and **the Principle of Factual Verification**, collectively enforce a critical data fidelity mandate. This protocol requires every piece of reported information to be explicitly present in and directly traceable to the source output, which fundamentally prevents speculative inaccuracies and ensures all subsequent automated decision-making is based on a verifiable, ground-truth foundation.

In addition,we have tailored specific prompts based on the characteristics of each environment to highlight potential details that may require attention, allowing LLMs to capture key information more quickly. For example, we design prompts to make the LLMs focus on combining physical absolute paths with exploitable SQL injection pathways into uploadable paths, facilitating Webshells uploads. This prompt design approach, while somewhat redundant, effectively directs the LLM's focus toward objectives in the current operational context, yielding highly effective results.

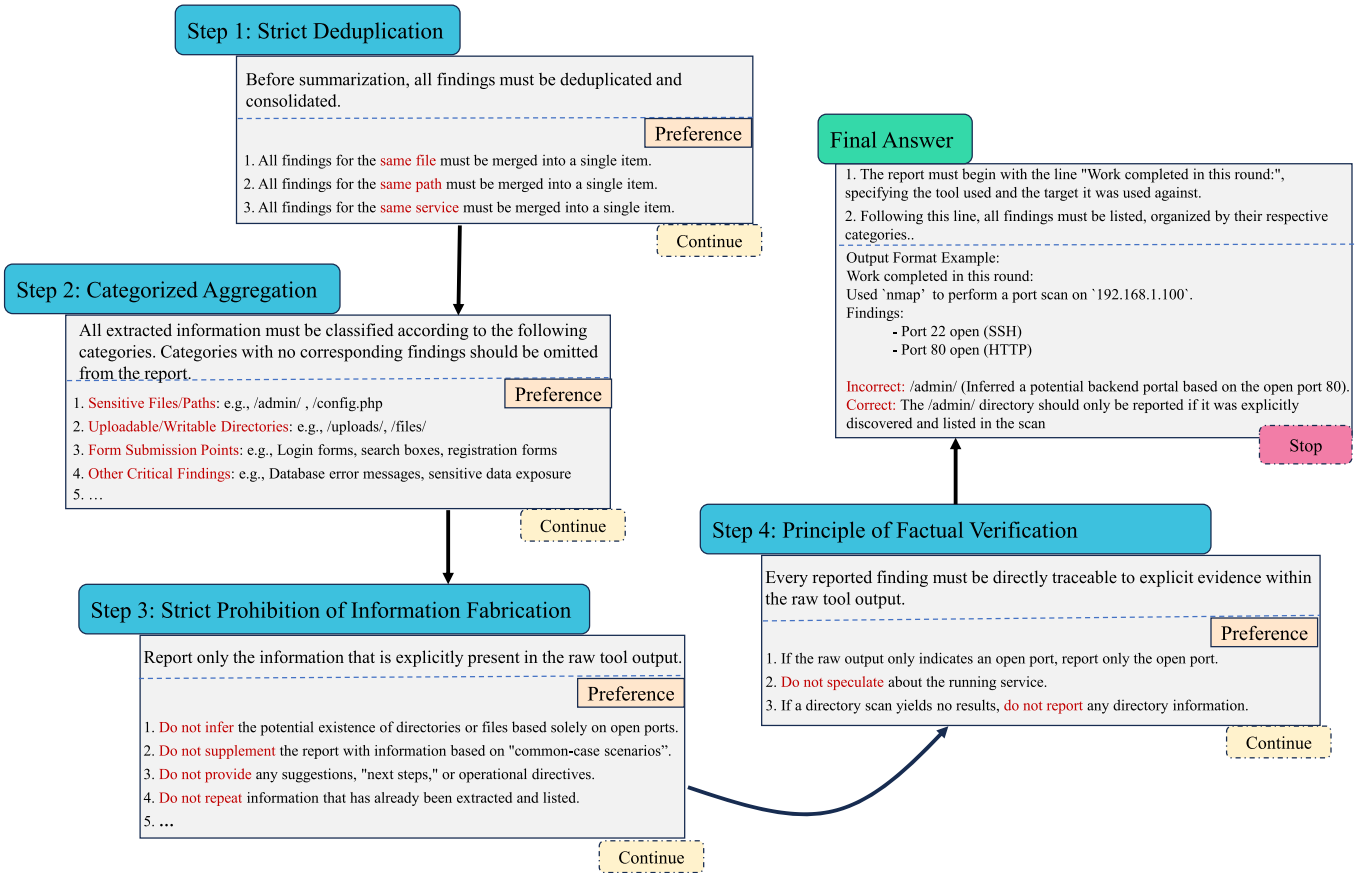


**Fig. 5.** The preference-based Chain-of-Thought Prompting.

## 5. Experiments

### 5.1. Environments setup

To validate the effectiveness of the proposed approach, we conducted three dedicated experiments: Task completion performance analysis and Reasoning similarity analysis, where the first series of experiments analyzed and evaluated the impact of different methods, LLMs, and core mechanisms. Six penetration testing environments were selected from the VulnHub [27] platform, which is a widely utilized platform for penetration testing, hosting a diverse repository of open-source vulnerable network environments. For convenience of representation, different environments are denoted as " env " , as detailed in the Table 4. The specific attack and exploitation techniques are presented in Table A.1.

The designed approaches employed GPT-4.1-mini [13] as the foundational LLM, and compared two LLMs: GPT-4o-mini [14] and Qwen 72B [15]. The objective is to obtain webshell execution privileges without human intervention, demonstrating end-to-end operational autonomy throughout the penetration testing workflow.

To mitigate inconsistent reasoning paths of large models in complex scenarios, each approach was executed 5 `times` in distinct network environments. Each `episode` represents a complete penetration testing task cycle - from target machine initialization to task completion (e.g., establishing webshell access or timeout). To accurately evaluate agent efficiency, we recorded episodes and steps per execution. An `episode` defines an agent's full task lifecycle. For instance, the period spanning from when the *ReconAgent* receives a task from the *MasterAgent* until it submits the final summary confirming task completion is considered an `episode`. A `step` refers to the number of commands executed within an `episode`, e.g., if *ReconAgent* runs 5 commands after receiving a task, that `episode` comprises 5 `steps`.

**Table 4**
Experimental environments.

| Index | Environments | Represented by |
|---|---|---|
| 1 | AI Web 1.0 [28] | env1 |
| 2 | from_sqli_to_shell_i386 [29] | env2 |
| 3 | JIS-CTF [30] | env3 |
| 4 | Metasploitable: 2 [31] | env4 |
| 5 | SickOs: 1.2 [32] | env5 |
| 6 | Basic Pentesting: 1 [33] | env6 |

Based on the statistics of `times`, `episodes`, and `steps`, we can define four metrics.

(1) **Penetration Success Rate (PSR)**: The proportion of successfully completed penetration testing tasks across all times.

$$\mathrm{PSR} = \frac{p}{n} \times 100\,\%$$

Where, $p$ is the number of times the proposed approach successfully obtained webshell privileges, and $n$ is the total number of times tested. The higher the PSR value, the stronger the task completion capability. A value of 100 % indicates successful automated webshell acquisition across all five test runs.

(2) **Average Episodes (AE)**: The average number of the episodes attempts in all times.

$$\mathrm{AE} = \frac{\sum_{i=1}^{n} e_i}{n} \times 100\,\%$$

**Table 5**
Comparative approaches in the task completion performance analysis.

| Index | Approach | Description |
|---|---|---|
| 1 | PTFusion | The approach metioned in this paper. |
| 2 | PTFusion with action history | only on the relies on the currently executed actions to guide task execution. |
| 3 | PTFusion with DKG | only on the relies on the DKGs to guide task execution. |
| 4 | PentestGPT with an experienced expert | A security expert executes operations based on the recommendations provided by the PentestGPT and its own experience. |
| 5 | PentestGPT with a beginner | A security beginner executes operations step-by-step solely based on the PentestGPT's recommendations. |

Where, $e_i$ represents the number of episodes executed in the $i$-th time. The Lower the AE value, the more accurate the strategic decisions of the *MasterAgent*.

(3) **Average Steps (AS)**: The average number of steps required to complete reconnaissance and attack tasks per time.

$$\mathrm{AS} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} s_{i,j}}{n} \times 100\,\%$$

Where, $s_{i,j}$ represents the number of steps executed in the $j$-th episode of the $i$-th time, $m$ is the number of episodes in $i$-th time. The lower the AS value, the stronger the capability of the *ReconAgent* and *AttackAgent* to execute tactical decisions. This enables them to more effectively overcome the multi-source data fusion and alignment challenges, leveraging the LLM to seamlessly transform information into tactical decision-making commands.

(4) Reasoning Similarity Score (RSS): This metric quantifies the similarity between reasoning step sequences across different episode runs within the same scenario, reflecting the stability and repeatability of the decision-making process.

$$\mathrm{RSS}_{i,j} = \frac{1}{1 + \mathrm{DTW}(X_i, X_j)}.$$

Where $X_i$ and $X_j$ denote the reasoning sequences of the $i$-th and $j$-th runs respectively, and DTW$(\cdot)$ represents the Dynamic Time Warping distance, with smaller distances indicating higher similarity. RSS values range from 0 to 1, with values closer to 1 indicating greater reasoning consistency. The DTW distance between two sequences

$$X = (x_1, x_2, \ldots, x_N), \quad Y = (y_1, y_2, \ldots, y_M)$$

is computed by minimizing the cumulative cost matrix $D$:

$$D(i,j) = \|x_i - y_j\| + \min\left(D(i-1,j), D(i,j-1), D(i-1,j-1)\right)$$

with boundary conditions:

$$D(0,0) = 0, \quad D(i,0) = D(0,j) = \infty \quad \text{for } i,j > 0.$$

Where $\|\cdot\|$ denotes a distance metric such as the Euclidean distance. DTW aligns sequences by optimally warping their time axes to minimize cumulative distance, enabling effective similarity measurement despite temporal distortions.

The RSS for each scenario is calculated as the average of all pairwise similarities:

$$\mathrm{RSS} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathrm{RSS}_{i,j}.$$

Where $n$ is the number of experimental runs in the scenario. A higher RSS indicates that the approach consistently follows similar reasoning patterns across trials, demonstrating robustness in decision-making and low variance in strategic execution.

All experiments were conducted on an Intel i7-12700K/32GB RAM/1TB SSD hardware configuration, and used virtual machines to construct the target network environment.

### 5.2. Task completion performance analysis

#### 5.2.1. Impact evaluation of different approaches

Considering that there are currently only two open-source LLM-driven penetration testing programs, PentestGPT and PentestAgent. However, PentestAgent is currently only used in some simple environments, it may only take 2–3 steps to complete vulnerability exploitation, so is not suitable for comparative analysis in more complex environments. Therefore, we compared PTFusion and its variants with two categories of PentestGPT approaches in this experiment, as detailed in the Table 5.

Fig. 6 compares the PSR metric of the five methods mentioned above.

As shown in Fig. 4, PTFusion achieved a 100 % penetration success rate (PSR) in all six environments. Its success depends not only on the collaboration of three agents, but is also facilitated by the preference-based chain-of-thought and context-aware dynamic knowledge graph. Especially the prompts specifically designed for each unique environment enable the *MasterAgent* to quickly grasp the current situation during task planning, thereby achieving near-perfect judgment accuracy.

In the relatively simple env2 and env6, both PTFusion and its variants achieved success. However, there may be differences due to varying levels of expertise among experts. In the env2, PentestGPT configured with a beginner-level operator fails more frequently because its generated SQL injection commands are often inaccurate. These commands require manual correction based on personal experience to execute accurately. Consequently, PentestGPT configured with an experienced expert consistently succeeds. While in the env6, a weak password can be used to login the website, and once PentestGPT with a security expert did not provide any reasonable suggestions. Even though the scenario is relatively simple, PentestGPT with a beginner has not generated executable payloads, indicating that it can currently only support strategic planning and cannot guide actual task execution.

Env3 represents a relatively common class of environments. Although the execution process involves numerous steps, the tasks progress in a predominantly sequential manner. Whether PentestGPT is operated by an experienced expert or a beginner, success primarily depends on identifying potential passwords in files prior to login for file uploads. Consequently, the task success rates remain consistently high.

In the env1, PTFusion with action history frequently fails to incorporate the previously obtained absolute physical path when constructing the webshell upload path due to insufficient environmental awareness. To form the full upload path, PTFusion requires combining the detected SQL injection path with the physical path, which demands both contextual awareness for capturing critical information and relatively accurate global planning. While PTFusion with action history only recorded past actions without situational awareness. This contextual blindness leads to oversight of critical details during strategic planning. In contrast, PTFusion with DKG does not record action history, often trapping itself in local action loops that derail execution direction. Both approaches may result in multiple potential failures during the PTFusion execution.

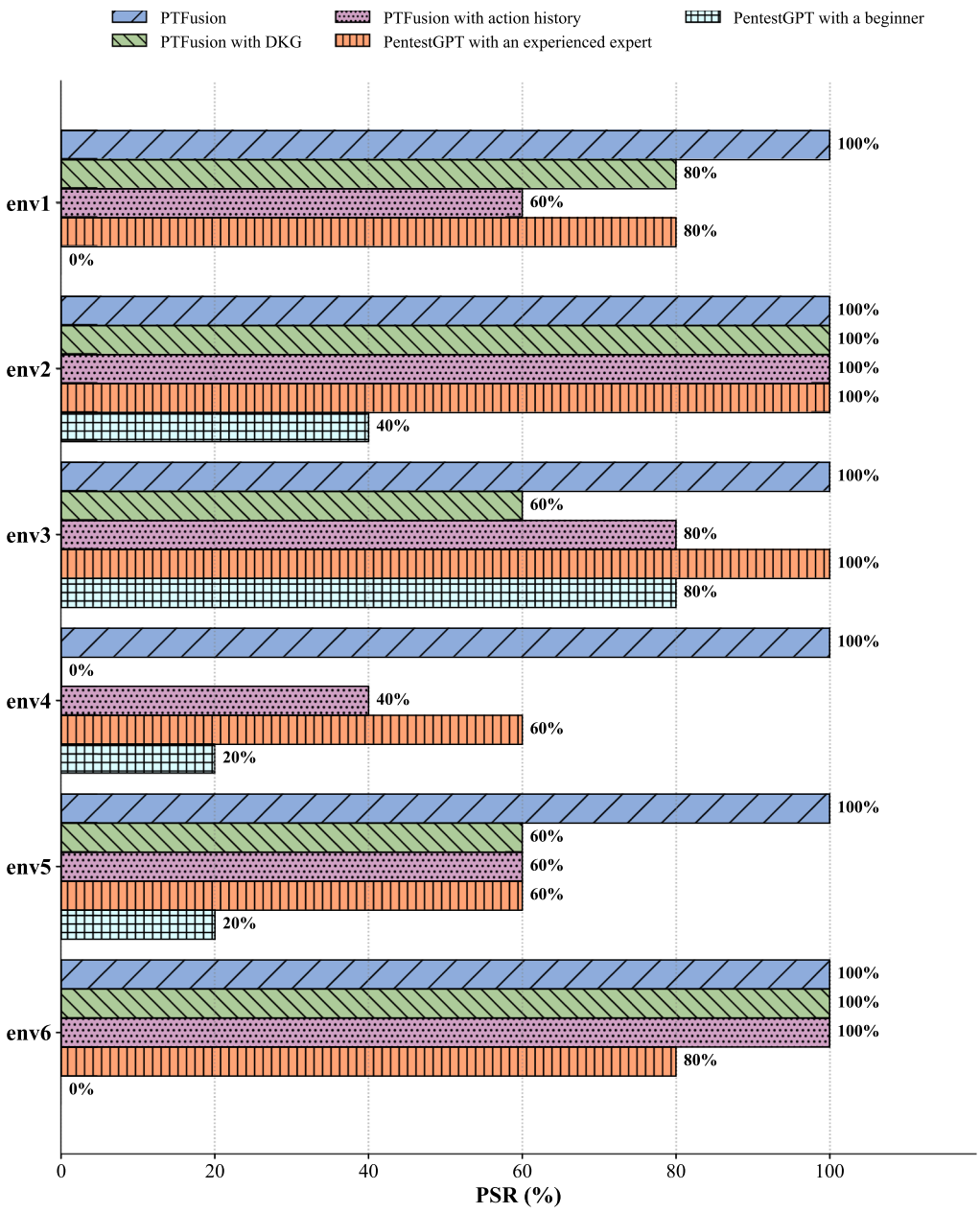


**Fig. 6.** Task completion performance analysis between different approaches.

The same scenario occurs in env4 and env5. In the env4, PTFusion with DKG completely fails due to excessive reliance on knowledge graph reasoning without sufficient adaptive feedback, resulting in repeated erroneous command execution. In contrast, PentestGPT with a security expert demonstrates moderate performance, though PentestGPT with a beginner struggled to meet the environment's strict command sequence requirements. The attack sequence in env5 requires PTFusion to perform WebDAV method discovery as a prerequisite condition for successful command injection and ultimate attack completion, so PTFusion with DKG and PTFusion with action history both drop to 60 % PSR. PentestGPT with an experienced expert manages to adapt partially, while a beginner frequently misidentifies valid exploitation paths.

Regarding task execution success rates and stability, PTFusion achieves near-perfect performance across virtually all scenarios. However, since the LLM receives varying information and reasoning paths in complex task environments, certain task volatility persists. In contrast to PentestGPT, which often requires security experts' experience to execute precise commands and contextual task operations in complex scenarios, PTFusion's integration of the dynamic knowledge graph and action history effectively balances automation robustness with operational accuracy in sophisticated web penetration testing, ensuring orderly and accurate task execution.

### 5.2.2. *Impact evaluation of different LLMs*

The second experiment analyzes the impact of different LLMs on PTFusion's performance. Fig. 7 presents the comparative results.

As evidenced in Fig. 7, the selection of LLMs exerts a substantial influence on PTFusion's performance. Although no literature or report has disclosed technical details of GPT-4.1-mini, it likely employs a Mixture-of-Experts (MoE) architecture similar to GPT-4o-mini, utilizing sparse activation to reduce computational costs. However, the difference is that GPT-4.1 mini has a context of 1M tokens, which is more than GPT-4o mini's 128K tokens, and there have been significant improvements in code generation and instruction compliance. These advantages directly

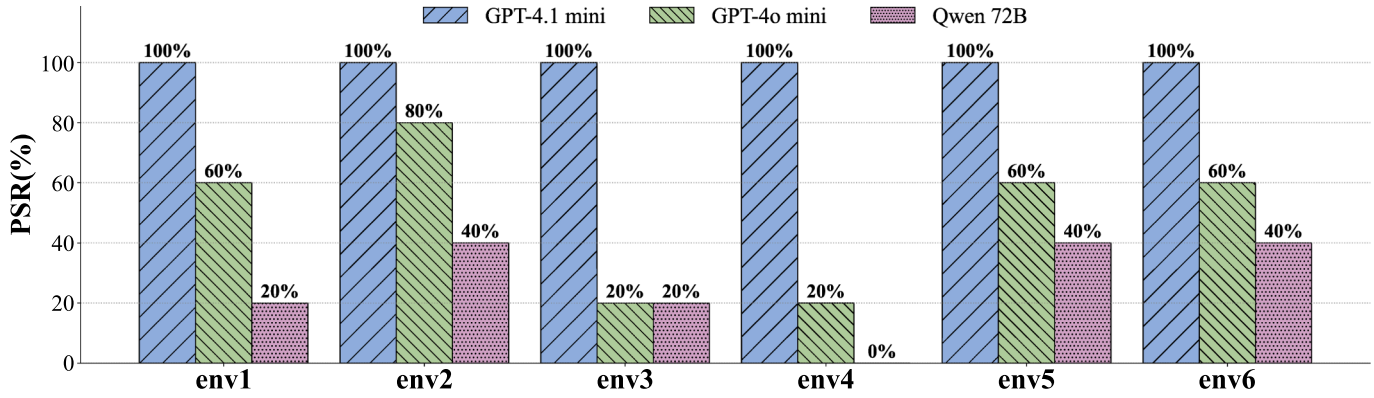


**Fig. 7.** Task completion performance analysis between different LLMs.

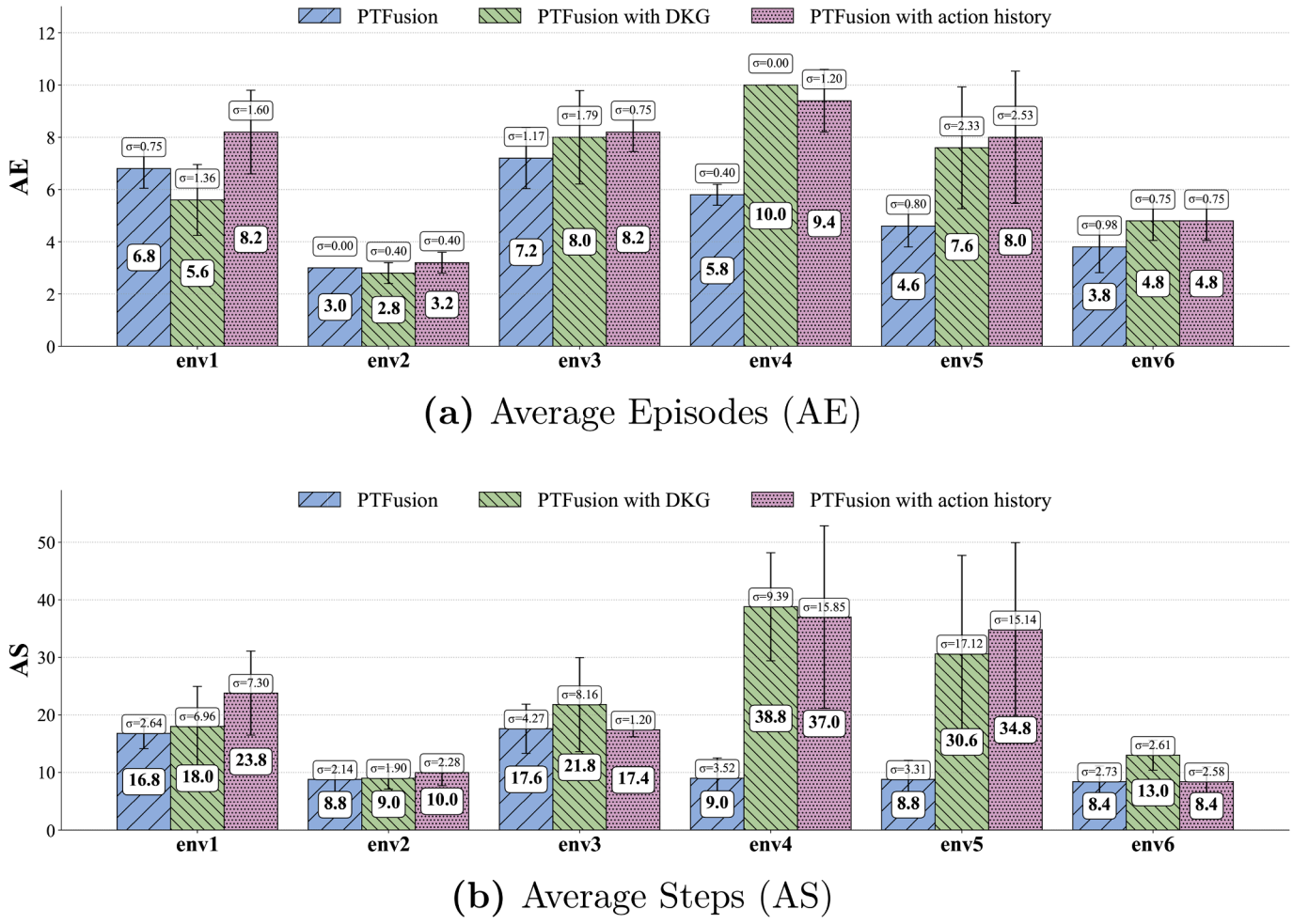


**Fig. 8.** Impact evaluation of the Context-Aware Knowledge Fusion Mechanism.

translate to penetration testing tasks, where its structured reasoning enables accurate attack-path deduction and deterministic command synthesis.

Conversely, GPT-4o mini has relatively limited performance. Especially in the third environment, agents need to extract information such as upload paths and PHPSESSION. However, it is often difficult to extract valuable knowledge from this information, and tends to generate "hallucinations". For example, it may generate wrong accounts and passwords to log in the administrator page, and notify that the login is correct. In fact, it does not complete the specified command and returned an incorrect result, so it is unable to execute the next task.

In the more challenging env4 and env5, the performance gap widens further: GPT-4.1 mini maintains 100 % PSR, where GPT-4o mini drops sharply to 20 % and 60 % respectively. Qwen 72B performs the worst, failing entirely in env4 and achieving only 40 % in env5. Even in env6, GPT-4.1 mini sustains perfect success, while GPT-4o mini and Qwen 72B reach only 60 % and 40 % respectively.

Among these six environments, QWB-72B demonstrates the poorest performance in web penetration testing tasks. This is because QWQ-72B struggles to simultaneously maintain both global comprehension capabilities and precise attack payload generation.

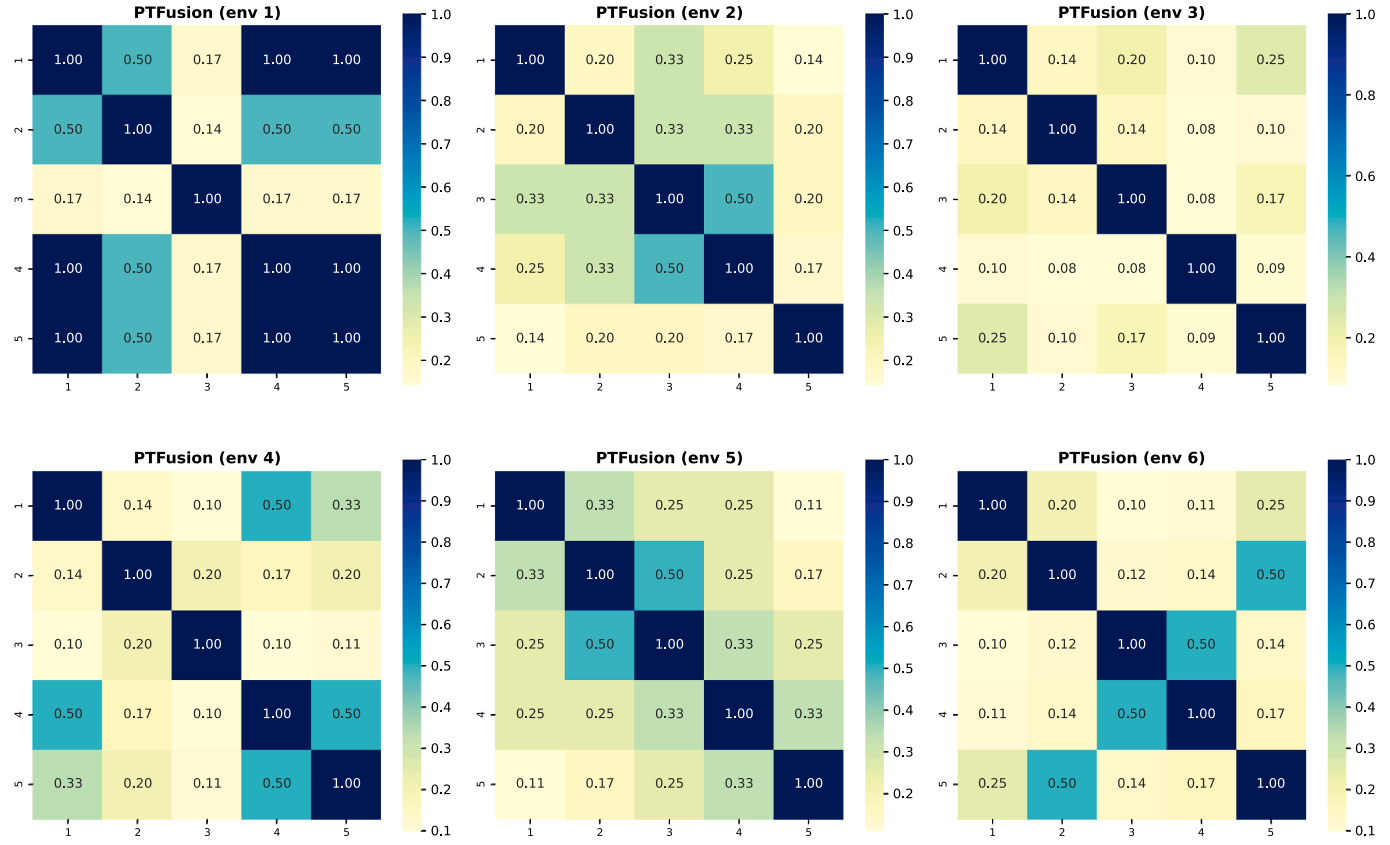


**Fig. 9.** Reasoning similarity heatmap of PTFusion.

*5.2.3. Impact evaluation of the Context-Aware Knowledge Fusion Mechanism*

In the third experiment, we further analyzed the impact of the Context-Aware Knowledge Fusion Mechanism on task execution in the PTFusion. Fig. 8 compares the changes in metrics AE and AS across three experimental environments.

Based on the experimental results, the three approaches show minimal differences in the relatively simple target environment env2.However, in the more complex env1 and env3, PTFusion with action history exhibits significantly more inter-agent interactions. This occurs because the method fails to analyze the target's attack surface through knowledge graphs, necessitating multiple rounds of confirmation to accurately comprehend the current task. In contrast, both PTFusion and PTFusion with DKGs leverage dynamic knowledge graphs to plan their respective tasks. Consequently, agents gain clearer understanding of their individual assignments and execute tactical tasks more effectively.

A similar trend is observed in env4 and env5, removing either the DKG or action history module is accompanied by notably higher AE and AS values, such as over 37 steps in env4 for PTFusion with action history and nearly 39 steps for PTFusion with DKG, as well as more than 30 steps in env5 for these reduced variants. In env6, all three methods achieve relatively low AS values, yet the full PTFusion still records the smallest AE, indicating quicker completion of task sequences.

Regarding the AE metric, PTFusion evidently completes penetration testing tasks across different environments with fewer steps. Combined with the PSR metric from the first experiment, these results demonstrate PTFusion's capability to effectively process multi-source data in complex environments, converting it into actionable knowledge to guide strategic planning and task execution. When the PTFusion lacks the DKG module, agents tend to fall into "repeated scan" or "local verification loops", such as repeatedly analyzing the same PHP file which results in decreased success rates and sharply increased operational steps. When the PTFusion relies solely on DKG, the absence of real-time feedback (e.g., tool execution failure signals) leads to rigid planning and limited strategy adjustments, consequently increasing operational steps.

Therefore, the complete PTFusion demonstrates the key advantages of collaborative integration between DKG and the action history. The absence of either component creates operational traps. For example, the PTFusion with action history cannot dynamically adjust strategies. This synergy not only enhances robustness against environmental uncertainties but also confirms the indispensable role of context-aware knowledge fusion in balancing exploration integrity and execution efficiency for automated penetration testing.

*5.3. Reasoning similarity analysis*

The Reasoning Similarity Score (RSS) quantifies the consistency of the reasoning steps across multiple experimental runs within the same scene, thus serving as an indicator of the stability and repeatability of the decision-making process. As defined, the RSS is derived from the pairwise Dynamic Time Warping (DTW) distances between reasoning sequences, with higher values signifying greater reasoning alignment. Fig. 9 presents the RSS matrices.

In the figure above, a higher RSS value indicates greater similarity in PTFusion's outputs across different inference rounds, while a lower value represents higher variability in its reasoning.

Env1 exhibits generally high RSS values, with several pairwise similarities at or near 1 (e.g., the first, fourth and fifth time), reflecting robust and consistent reasoning strategies under relatively stable conditions. The mean RSS for env1 approximates 0.61, indicating strong reasoning stability.

Although env3 represents a sequentially structured environment, PTFusion requires the highest number of execution episodes (mean = 7.2)

among all tested scenarios. This indicates repeated strategic replanning during task execution, consequently resulting in greater variability in its reasoning paths.

The tested environments show differential reasoning consistency patterns, where env2, env4, env5 and env6 maintain moderate strategic alignment as evidenced by Residual Similarity Scores (RSS) centered around 0.33. This indicates progressively more heterogeneous reasoning patterns that likely reflect increasing environmental complexity, greater behavioral variability, or higher uncertainty in decision-making scenarios. This spectrum of similarity scores suggests a clear trend where more challenging environments lead to greater divergence in strategic reasoning and execution pathways.

Overall, the data reveal that reasoning similarity is highest in simpler or more constrained scenes, while more challenging environments elicit greater divergence in reasoning sequences.

### 5.4. Discussions

**Stable and excellent performance.** The experimental results comprehensively demonstrate the effectiveness and robustness of the proposed PTFusion framework in automated penetration testing. Compared with baseline approaches, including PentestGPT variants operated by both security experts and beginners, PTFusion consistently achieves superior penetration success rates and demonstrates strong adaptability across diverse and complex environments.

Reasoning similarity analysis also proves that PTFusion maintains consistent decision-making patterns in simpler scenes, while increased environmental complexity naturally induces more diverse reasoning trajectories. This variability underscores the challenges inherent in autonomous penetration testing within dynamic, real-world settings.

**Significant impact of the Context aware Knowledge Fusion Mechanism.** The integration of context-aware dynamic knowledge graphs and action history proves critical in enabling multi-agent coordination, precise strategic planning, and dynamic adjustments during task execution. Variants of PTFusion lacking either component show marked performance degradation, highlighting the indispensability of this synergistic knowledge fusion mechanism.

**The reasoning and instruction generation capabilities of LLMs is also important.** The choice of underlying LLM models significantly impacts performance. LLMs with advanced architectures, such as GPT-4.1 mini, offer more accurate reasoning and command synthesis, thereby maintaining high success rates even in challenging scenarios. In contrast, smaller or less capable models exhibit increased error rates and hallucinations, especially under complex task demands.

In summary, the experimental evidence validates PTFusion's ability to balance automation robustness with operational accuracy, positioning it as a promising approach for sophisticated and reliable autonomous penetration testing.

## 6. Related work

### 6.1. RL-based penetration testing

Penetration testing (PT) is a widely used method for evaluating network security by simulating real-world attacks. Traditional approaches rely heavily on human expertise, making them costly, time-consuming, and difficult to scale for complex infrastructures. Reinforcement learning (RL) offers an alternative by enabling agents to autonomously plan and execute multi-step attacks through adaptive sequential decision-making [34]. Nevertheless, its application faces significant challenges, including vast state-action spaces, low sample efficiency, limited policy generalization, and the realism gap between simulated training environments and real-world deployments [35].

To address these issues, researchers have proposed hierarchical RL frameworks to decompose PT tasks into manageable subtasks. Li et al. [1] designed a hierarchical deep RL model that integrates expert prior knowledge, using layered agents and knowledge graph-driven action constraints to improve exploration efficiency in large-scale network scenarios. Tran et al. [5] introduced HA-DRL, which employs algebraic action decomposition to handle exponentially growing action spaces, enabling faster convergence than conventional deep Q-learning agents. Similarly, Ghanem et al. [6] treated PT as a partially observed Markov decision process (POMDP) and leveraged model-based RL to identify optimal attack policies. Liu et al. [36] further enhanced this approach by introducing network-level and host-level agents with action masking and invalid action discrimination, improving stability and efficiency.

Building on hierarchical architectures, action embedding techniques have been developed to address large discrete action spaces. Nguyen and Uehara [7] proposed a hierarchical action embedding informed by the MITRE ATT&CK knowledge base, capturing semantic relations between actions to guide agent decision-making in complex networks. Zhou et al. [2] advanced this idea with APRIL, which embeds discrete actions into a continuous semantic space and applies exploration refinement strategies to improve scalability and cross-scenario transfer. Multiple-level and multilayer action representations [8,9] have also demonstrated improved accuracy and learning performance across varied network complexities.

Another line of research integrates imitation learning to accelerate convergence and incorporate expert strategies. Wang et al. [10] proposed DQfD-AIPT, which combines deep Q-learning from demonstrations with structured expert knowledge to reduce overfitting and improve policy robustness, particularly in environments containing honeypots. Chen et al. [3] introduced GAIL-PT, which uses generative adversarial imitation learning to guide RL agents through expert-derived state-action pairs, achieving superior performance over existing baselines in both simulated and real network settings.

To improve adaptability in dynamic environments, meta-RL approaches have been explored. Zhou et al. [37] developed GAP, a Real-to-Sim-to-Real framework that leverages domain randomization and meta-RL to train generalizable policies capable of zero-shot transfer and rapid adaptation to unseen network scenarios.

Finally, advances in training environments aim to reduce the simulation-to-reality gap. Nguyen et al. [4] introduced PenGym, a realistic PT environment that supports genuine attack actions and automated network creation, leading to higher success rates and stability in real infrastructures compared to purely simulated platforms.

Despite these advances, RL-based PT still requires extensive interaction for policy learning, remains sensitive to environmental changes, and struggles with orchestrating heterogeneous security tools-motivating the exploration of complementary paradigms.

### 6.2. LLM-enhanced penetration stesting

Recent advances in large language models (LLMs) offer a complementary direction for penetration testing automation, shifting the focus from pure policy learning toward intelligent tool orchestration, contextual reasoning, and knowledge integration. Unlike RL agents, which acquire strategies through trial-and-error interactions with an environment, LLM-powered penetration testing systems leverage extensive pretrained knowledge, natural language understanding, and flexible reasoning capabilities to interpret tool outputs, plan attack steps, and adapt to diverse environments without explicit retraining [38].

Several studies have investigated the application of LLMs within penetration testing workflows. Hilario et al. [39] systematically examined the role of generative AI across the five classic stages of penetration testing, demonstrating that ChatGPT 3.5 can accelerate task execution, enhance report quality, and inspire creative scenario design, while also cautioning about risks such as unintended consequences and lack of control. Pratama et al. [40] introduced CIPHER, an LLM-based chatbot designed to assist ethical security researchers and lower the entry barrier for novice practitioners. Zhong et al. [41] developed PenQA, a curated instructional dataset combining theoretical and practical penetration

**Table A.1**
Per-environment exploitation steps and descriptions.

| Environments | Technique | Descriptions |
|---|---|---|
| AI Web 1.0 | Port Scanning | Detected TCP port 80 open (HTTP/Apache). |
| | Web Enumeration | robots.txt disclosed /m3diNf0/ and /se3reTdir777/uploads/. |
| | Information Disclosure | /m3diNf0/info.php revealed DOCUMENT_ROOT = /home/www/HTML/…. |
| | Input Point Identification | /se3reTdir777/index.php contains a POST parameter uid. |
| | SQL Injection | Confirmed SQL injection on parameter `uid`. |
| | Shell Construction | Created a local one-liner PHP webshell file `hack.php`. |
| | File Write via SQL Injection | Wrote `hack.php` to `/se3reTdir777/uploads/hack.php`; remote write confirmed. |
| | Success Criterion | Remote webshell write confirmed; success condition met. |
| From SQL to Shell | Port Scanning | Detected TCP ports 22 (SSH/OpenSSH 5.5p1) and 80 (HTTP/Apache 2.2.16 Debian) open. |
| | Web Enumeration | Site structure shows `/admin/` and `cat.php?id=1/2/3`; page assets indicate `/admin/uploads/`. |
| | SQL Injection | Confirmed SQL injection on `cat.php?id=3` (parameter `id`). |
| | Database Discovery | Detected databases `photoblog` (application DB) and `information_schema`. |
| | Table Enumeration | In database `photoblog`: tables `categories`, `pictures`, `users`. |
| | Column Enumeration | In `photoblog.users`: columns `id`, `login`, `password`. |
| | Credential Dump | Dumped admin's MD5 hash and cracked password to `P4ssw0rd`. |
| | Success Criterion | Recovery of admin credentials satisfies the experiment's success condition; login not required by protocol. |
| JIS-CTF | Port Scanning | Detected TCP ports 22 (SSH) and 80 (HTTP) open. |
| | Web Enumeration | `login.php` is the login page; `robots.txt` disclosed `/admin_area/`, `/uploads/`, `/uploaded_files/`. |
| | Information Disclosure | HTML comments in `/admin_area/index.php` contain credentials `admin:3v1l_H@ck3r`. |
| | Authentication | Logged in with the credentials and obtained a valid session for the Upload Center. |
| | Upload Functionality Verification | The Upload Center form uses field name `image`; test upload returned "Success". |
| | Shell Construction | Created a local PHP one-liner webshell file `shell.php`. |
| | Remote File Upload | Uploaded `shell.php` via the authenticated upload form. |
| | Webshell Verification | Accessed `/uploaded_files/webshell.php`; presence confirmed (HTTP 500 observed). |
| Metasploitable 2 | Port Scanning | Detected multiple services, including 2121 (FTP/ProFTPD 1.3.1), 21 (FTP/vsftpd 2.3.4), and 8180 (HTTP/Apache Tomcat). |
| | FTP Authentication | Authenticated to the FTP service on port 2121 using default credentials `msfadmin:msfadmin`; the FTP root directory listing revealed `vulnerable`. |
| | FTP Enumeration | Confirmed write permission on `/vulnerable/`; observed subdirectories `mysql-ssl`, `samba`, `tikiwiki`, `twiki20030201`. |
| | Shell Construction | Constructed a local PHP one-liner webshell `webshell.php`. |
| | Remote Upload via FTP | Uploaded `webshell.php` to `/vulnerable/`; transfer completed successfully. |
| | Webshell Verification | Retrieved `/vulnerable/webshell.php` via FTP to confirm file presence. |
| SickOS 1.2 | Port Scanning | Detected TCP ports 22 (SSH) and 80 (HTTP) open. |
| | HTTP Method Enumeration | On `/test/`, HTTP OPTIONS shows `Allow` includes `PUT` and WebDAV methods (e.g., PROPFIND, MKCOL, LOCK/UNLOCK). |
| | Shell Construction | Created a local PHP webshell file `shell.php`. |
| | Remote File Upload | Uploaded `shell.php` to `/test/shell.php` via HTTP PUT; server acknowledged creation. |
| | Webshell Verification | Accessed `/test/shell.php?cmd=whoami`; HTTP 500 observed, confirming file presence. |
| Basic Pentesting 1 | Port Scanning | Detected TCP ports 21 (FTP), 22 (SSH), and 80 (HTTP) open. |
| | Web Enumeration | Discovered `/secret/` and WordPress endpoints `/secret/wp-admin/` and `/secret/wp-login.php`. |
| | Authentication | Logged into WordPress admin at `/secret/wp-admin` with `admin:admin`; obtained an authenticated session. |
| | Post-login Access Confirmation | Backend features accessible under the authenticated session (success criterion met). |

testing knowledge to improve LLM reasoning and procedural guidance. Bianou and Batogna [42] proposed PENTEST-AI, a multi-agent framework integrating the MITRE ATT&CK knowledge base with LLM agents to automate adversarial emulation workflows.

Other works have emphasized multi-agent orchestration and integration with external intelligence sources. PentestAgent [18] employs a hierarchical multi-agent framework for reconnaissance, search, planning, and exploitation, augmented by retrieval-augmented generation (RAG) to incorporate up-to-date vulnerability intelligence. PentestGPT [17] adopts a human-in-the-loop architecture, combining a high-level reasoning module, an action generation module, and a parsing module to condense verbose outputs into actionable intelligence. RedTeamLLM [43] further extends the agentic AI paradigm with explicit modules for summarizing, reasoning, and acting, tackling challenges such as plan correction, memory management, and context window constraints.

Practical system implementations have demonstrated LLMs' adaptability to real tools and scenarios. Salem and Mrian [44] integrated LLMs with Metasploit for autonomous vulnerability analysis, exploit selection, and payload customization, while Henke [45] developed AutoPentest, a GPT-4o-based agent system capable of executing multi-step black-box penetration tests with external tool augmentation. Beyond single-agent setups, Caturano et al. [46] showcased how two LLMs with complementary expertise can collaboratively generate working exploits through structured dialogue.

These LLM-enhanced approaches represent a paradigm shift from end-to-end policy learning toward reasoning-centric orchestration, dynamic multi-tool coordination, and specialized knowledge integration. By bridging pretrained intelligence with operational flexibility, they open new avenues for automating penetration testing in complex and evolving environments, while raising new challenges in controllability, safety, and ethical deployment.

## 7. Conclusions

This study presents PTFusion, a semi-decentralized multi-agent framework that fundamentally advances autonomous penetration testing by integrating Large Language Models (LLMs) with dynamic knowledge orchestration. By leveraging the Model Context Protocol (MCP), our system overcomes critical limitations of prior reinforcement learning (RL) and human-in-the-loop LLM approaches and employs a context-aware knowledge fusion mechanism to ensure verifiable intelligence integrity for reliable decision-making. Empirical validations confirm that PTFusion consistently achieves autonomous, stable performance across diverse web environments, outperforming experience-dependent systems like PentestGPT. This work establishes a new paradigm for intelligent security automation, with future research directions focusing on cross-segment network penetration testing and architectural optimization through integration of a wider range of penetration testing tools.

## CRediT authorship contribution statement

**Wenhao Wang:** Writing – original draft, Methodology, Investigation, Funding acquisition, Conceptualization; **Hao Gu:** Validation, Software; **Zhixuan Wu:** Visualization, Validation; **Hao Chen:** Visualization, Validation; **Xingguo Chen:** Writing – review & editing, Data curation; **Fan Shi:** Writing – review & editing, Funding acquisition.

## Data availability

Data will be made available on request.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper. The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

## Acknowledgment

We acknowledge funding from the China Postdoctoral Science Foundation under Grant No. 2024M764341, the National Natural Science Foundation of China (Grant No. 62276142), and support from the Science and Technology on Information Systems Engineering Laboratory.

## Appendix A. Optimal exploitation paths for each experimental environment

This appendix presents the optimal exploitation paths for all experimental environments, distilled from validated execution logs and expressed as technique-description pairs. For each environment, we report only the steps that constitute the shortest verified path to success (Table A.1).

## References

[1] Q. Li, M. Zhang, Y. Shen, et al., A hierarchical deep reinforcement learning model with expert prior knowledge for intelligent penetration testing, Comput. Secur. 132 (2023) 103358. https://doi.org/10.1016/j.cose.2023.103358
[2] S. Zhou, J. Liu, Y. Lu, et al., APRIL: Towards scalable and transferable autonomous penetration testing in large action space via action embedding, IEEE Trans. Depend. Secure Comput. 22 (3) (2025) 2443–2459. https://doi.org/10.1109/TDSC.2024.3518500
[3] J. Chen, S. Hu, H. Zheng, et al., GAIL-PT: An intelligent penetration testing framework with generative adversarial imitation learning, Comput. Secur. 126 (2023) 103055. https://doi.org/10.1016/j.cose.2022.103055
[4] H.P.T. Nguyen, K. Hasegawa, K. Fukushima, et al., PenGym: realistic training environment for reinforcement learning pentesting agents, Comput. Secur. 148 (2025) 104140. https://doi.org/10.1016/j.cose.2024.104140
[5] K. Tran, A. Akella, M. Standen, J. Kim, D. Bowman, T. Richer, C.-T. Lin, Deep hierarchical reinforcement agents for automated penetration testing, 2021. arXiv preprint arXiv:2109.06449.
[6] M.C. Ghanem, T.M. Chen, E.G. Nepomuceno, Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks, J. Intell. Inf. Syst. 60 (2) (2023) 281–303.
[7] H.V. Nguyen, T. Uehara, Hierarchical action embedding for effective autonomous penetration testing, in: 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C), IEEE, 2022, pp. 152–157. https://doi.org/10.1109/QRS-C57518.2022.00030
[8] H.V. Nguyen, H.N. Nguyen, T. Uehara, Multiple level action embedding for penetration testing, in: Proceedings of the 4th International Conference on Future Networks and Distributed Systems, ICFNDS '20, Association for Computing Machinery, New York, NY, USA, 2021, pp. 53:1–53:9. https://doi.org/10.1145/3440749.3442660
[9] H.V. Nguyen, T. Uehara, Multilayer action representation based on MITRE ATT&CK for automated penetration testing, J. Inform. Process. 31 (2023) 562–577. https://doi.org/10.2197/ipsjjip.31.562
[10] Y. Wang, Y. Li, X. Xiong, J. Zhang, Q. Yao, C. Shen, DQfD-AIPT: an intelligent penetration testing framework incorporating expert demonstration data, Secur. Commun. Netw. 2023 (2023) 5834434. https://doi.org/10.1155/2023/5834434
[11] Z. Li, Y. Gao, G. Yuan, et al., CDME: convolutional dictionary iterative model for pansharpening with mixture of experts, IEEE Geosci. Remote Sens. Lett. 22 (2025). https://doi.org/10.1109/LGRS.2025.3545472
[12] A. Alboody, R. Slama, Graph transformer mixture-of-experts (GTMoE) for 3D hand gesture recognition, in: Intelligent Systems and Applications, Vol. 3, INTELLISYS 2024, 1067 of *Lecture Notes in Networks and Systems*, Springer, 2024, pp. 317–336. https://doi.org/10.1007/978-3-031-66431-1_21
[13] OpenAI, GPT-4.1-mini, 2024, https://platform.openai.com/.
[14] OpenAI, GPT-4o-mini, 2024, https://platform.openai.com/.
[15] J. Bai, S. Bai, Y. Chu, et al., Qwen Technical Report, 2023. arXiv preprint arXiv:2309.16609.
[16] C. Li, B. Gu, Z. Zhao, Y. Qu, G. Xin, J. Huo, L. Gao, Federated transfer learning for on-Device LLMs efficient fine tuning optimization, Big Data Min. Analyt. 8 (2) (2025) 430–446. https://doi.org/10.26599/BDMA.2024.9020068
[17] G. Deng, Y. Liu, V. Mayoral-Vilches, et al., PentestGPT: Evaluating and harnessing large language models for automated penetration testing, in: Proceedings of the 33rd USENIX Security Symposium, USENIX Association, Philadelphia, PA, USA, 2024, pp. 847–864.
[18] X. Shen, L. Wang, Z. Li, et al., PentestAgent: Incorporating LLM Agents to Automated Penetration Testing,(2024). arXiv preprint arXiv:2411.05185.
[19] Anthropic, Model Context Protocol Specification (Version 2024-11-05), 2024. https://modelcontextprotocol.io/specification.
[20] H. Liu, S. Zhou, C. Chen, et al., Dynamic knowledge graph reasoning based on deep reinforcement learning, Knowl. Base. Syst. 241 (2022) 108235.
[21] J.T. Aparicio, E. Arsenio, F. Santos, R. Henriques, Using dynamic knowledge graphs to detect emerging communities of knowledge, Knowl. Base. Syst. 294 (2024) 111671. https://doi.org/10.1016/j.knosys.2024.111671
[22] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, D. Chalef, Zep: a temporal knowledge graph architecture for agent memory,(2025). arXiv preprint arXiv:2501.13956.
[23] J. Ma, Z. Gao, Q. Chai, W. Sun, P. Wang, H. Pei, J. Tao, L. Song, J. Liu, C. Zhang, L. Cui, Debate on graph: a flexible and reliable reasoning framework for large language models, in: Proceedings of the AAAI Conference on Artificial Intelligence, 39, AAAI Press, 2025, pp. 24768–24776. https://doi.org/10.1609/aaai.v39i23.34658
[24] Y. Zhou, Z. Wang, Y. Liu, A local differential privacy hybrid data clustering iterative algorithm for edge computing, Chin. J. Electron. 33 (6) (2024) 1421–1434. https://doi.org/10.23919/cje.2023.00.332
[25] J. Ma, P. Wang, D. Kong, Z. Wang, J. Liu, H. Pei, J. Zhao, Robust visual question answering: datasets, methods, and future challenges, IEEE Trans. Pattern Anal. Mach. Intell. 46 (8) (2024) 5575–5594. https://doi.org/10.1109/TPAMI.2024.3366154
[26] J. Ma, Z. Gao, Q. Chai, J. Liu, P. Wang, J. Tao, Z. Su, FortisAVQA and MAVEN: a Benchmark Dataset and Debiasing Framework for Robust Multimodal Reasoning,(2025). arXiv preprint arXiv:2504.00487.
[27] VulnHub, VulnHub, 2024. https://www.vulnhub.com/.
[28] VulnHub, AI-Web-1.0, 2019. https://www.vulnhub.com/entry/ai-web-1,353/.
[29] VulnHub, from_sqli_to_shell_i386, 2012. https://www.vulnhub.com/entry/pentester-lab-from-sql-injection-to-shell,80/.
[30] VulnHub, JIS-CTF, 2018. https://www.vulnhub.com/entry/jis-ctf-vulnupload,228/.
[31] VulnHub, Metasploitable: 2,2012. https://www.vulnhub.com/entry/metasploitable-2,29/.
[32] VulnHub, SickOs: 1.2,2016. https://www.vulnhub.com/entry/sickos-12,144/.
[33] VulnHub, Basic Pentesting: 1, 2017. https://www.vulnhub.com/entry/basic-pentesting-1,216/.
[34] M.C. Ghanem, T.M. Chen, Reinforcement learning for efficient network penetration testing, Information 11 (1) (2020) 6. https://doi.org/10.3390/info11010006
[35] W. Wang, D. Sun, F. Jiang, X. Chen, C. Zhu, Research and challenges of reinforcement learning in cyber defense decision-making for intranet security, Algorithms 15 (4) (2022) 134. https://doi.org/10.3390/a15040134
[36] H. Liu, C. Liu, X. Wu, Y. Qu, H. Liu, An automated penetration testing framework based on hierarchical reinforcement learning, Electron. (Basel) 13 (2024) 4311. https://doi.org/10.3390/electronics13214311
[37] S. Zhou, J. Liu, Y. Lu, J. Yang, Y. Zhang, J. Chen, Mind the Gap: Towards Generalizable Autonomous Penetration Testing via Domain Randomization and Meta-Reinforcement Learning,(2025). arXiv preprint arXiv:2412.04078.
[38] M. Xu, J. Fan, X. Huang, C. Zhou, J. Kang, D. Niyato, S. Mao, Z. Han, X. Shen, K.-Y. Lam, Forewarned is Forearmed: A Survey on Large Language Model-based Agents in Autonomous Cyberattacks,(2025). arXiv preprint arXiv:2505.12786.
[39] E. Hilario, S. Azam, J. Sundaram, et al., Generative AI for pentesting: the good, the bad, the ugly, Int. J. Inf. Secur. 23 (3) (2024) 2075–2097. https://doi.org/10.1007/s10207-024-00835-x
[40] D. Pratama, N. Suryanto, A.A. Adiputra, et al., CIPHER: cybersecurity sintelligent penetration-testing helper for ethical researcher, Sensors 24 (21) (2024). https://doi.org/10.3390/s24216878
[41] X. Zhong, Y. Zhang, J. Liu, PenQA: a comprehensive instructionaldataset for enhancing penetration testing capabilities in language models, Appl. Sci. 15 (4) (2025). https://doi.org/10.3390/app15042117
[42] S.G. Bianou, R.G. Batogna, PENTEST-AI, an LLM-powered multi-agents framework for penetration testing automation leveraging MITRE ATTACK, in: 2024 IEEE International Conference on Cyber Security and Resilience, CSR, IEEE, IEEE, London, ENGLAND, 2024, pp. 763–770. 4th IEEE Annual International Conference on Cyber Security and Resilience (IEEE CSR), SEP 02–04, 2024. https://doi.org/10.1109/CSR61664.2024.10679480

[43] B. Challita, P. Parrend, RedTeamLLM: an Agentic AI framework for offensive security,(2025). arXiv preprint arXiv:2505.06913.
[44] M. Salem, M. Mrian, AI-driven penetration testing: automating exploits with LLMs and metasploit-A VSFTPD case study, in: 2025 International Conference on New Trends in Computing Sciences (ICTCS), IEEE, 2025, pp. 89–96. https://doi.org/10.1109/ICTCS65341.2025.10989363
[45] J. Henke, AutoPentest: Enhancing Vulnerability Management With Autonomous LLM Agents, (2025). arXiv preprint arXiv:2505.10321.
[46] F. Caturano, J. Ciotola, S.P. Romano, M. Varlese, A chit-chat between Lama 2 and ChatGPT for the automated creation of exploits, Comput. Netw. 270 (2025)111501. https://doi.org/10.1016/j.comnet.2025.111501