ITC 3/55
Information Technology and Control
Vol. 55 / No. 3/ 2026
pp. 1135-1155
DOI 10.5755/j01.itc.55.3.44840

APTMimic: Automated Attack Simulation for Dynamic APT Attack Sample Generation

Received 2026/04/22 | Accepted after revision 2026/06/29

HOW TO CITE: Zheng, X., Weng, Z., Zheng, J., Zhu, T., Li, J., Chen, B., Yuan, Q., Xia, Z. (2026) APTMimic: Automated Attack Simulation for Dynamic APT Attack Sample Generation. *Information Technology and Control*, 55(3), 1134-1154. https://doi.org/10.5755/j01.itc.55.3.44840

APTMimic: Automated Attack Simulation for Dynamic APT Attack Sample Generation

Xiangyang Zheng, Zhengqiu Weng*
School of Data Science and Artificial Intelligence, Wenzhou University of Technology, Wenzhou, 325000, China; e-mails: xyz@wzut.edu.cn (X. Z.), derisweng@wzut.edu.cn (Z. W.)

Jie Zheng, Tiantian Zhu, Qixuan Yuan
School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China; e-mails: zjutzj@zjut.edu.cn (J. Z.), ttzhu@zjut.edu.cn (T. Z.), zjutyqx@zjut.edu.cn (Q. Y.)

Jie Li, Bingrong Chen
School of Mathematics and Computer Science, Yan'an University, Yan'an 716000, China; e-mails: yandalijie@stu.yau.edu.cn (J. L.), chenbingrong@stu.yau.edu.cn (B. C.)

Zhiliang Xia*
Wenzhou Polytechnic, Wenzhou, 325000, China; e-mail: 2011041047@wzpt.edu.cn (Z. X.)

*Zhengqiu Weng and Zhiliang Xia contribute the same to the article and are the corresponding authors.

Corresponding author: derisweng@wzut.edu.cn (Z.W.), 2011041047@wzpt.edu.cn (Z.X.)

Advanced Persistent Threat (APT) attacks cause major losses across critical industries. Existing machine-learning detectors, restricted by the scarcity of provenance-level samples, rely on benign-behavior anomaly detection and suffer from high false-alarm rates and weak interpretability. We propose APTMimic, an automated APT-sample generation system for supervised provenance-graph research in authorized, isolated laboratory environments. APTMimic defines atomic attack behaviors and assembles a multi-source library; extracts attack-technique chains from CTI reports using a customized prompt framework with an LLM; organizes candidate chains via cost- and novelty-aware optimization; and generates attack code, simulates execution, and converts logs into provenance-graph samples. Experiments show APTMimic effec-

tively organizes atomic behaviors, extracts TTP sequences with high precision, and executes attack tasks reliably. The resulting samples expose clear weaknesses in four representative detection models, providing stress-testing resources for provenance-graph-based APT detection.

KEYWORDS: Advanced Persistent Threat (APT), APT sample, Attack graph/Attack chain graph, Graph generation, Endpoint threat detection, Large Language Model

1. Introduction

Advanced Persistent Threat (APT) attacks are continuous, well-organized attack activities launched by adversarial organizations against specific targets [1]. They are characterized by highly targeted objectives (typically government, defense, or finance), strong organization with clear division of labor, long dormancy spanning months to years, and high covertness in bypassing conventional defenses through complex multi-step techniques.

To detect APT attacks, security analysts parse system logs into a directed acyclic graph of entities and events, called the provenance graph [36], [31], which is the canonical data structure for APT analysis. Traditional rule-based detection methods [22], [16] either manually define detection rules from past experience or propagate predefined labels through the system [30], [22]. These methods achieve high accuracy on known patterns, but heuristic rules cannot keep pace with evolving APT techniques, yielding both false negatives and false positives.

Machine learning technology, with its ability to mine the intrinsic features of data, has shown significant advantages in supervised security tasks with ample training data, such as malware identification [3], [12] and malicious traffic detection [17], [2], making supervised cybersecurity detection techniques based on machine learning promising.

However, the scarcity of APT-attack samples severely limits supervised machine learning. Researchers have turned to anomaly detection [14, 20, 28, 36] to identify anomalous nodes, edges, or subgraphs by learning benign behavior. Yet anomaly-based detection has two issues: the boundary between anomalous and malicious behavior is increasingly blurred, producing high false-positive rates and low interpretability; and anomalies are not equivalent to attacks [9]. Manual collection via sandboxes [35] or endpoint probes [10], [7] is costly and quality-uncertain, motivating the following two challenges.

(C1) Limited sample generation diversity. Given the constantly changing landscape of cyberattacks, existing methods which reproduce attack behaviors based on known attack patterns to obtain samples, struggle to achieve large-scale sample production. As a result, they cannot ensure the diversity of the samples.

(C2) Low degree of attack simulation automation. Considering the limited manpower of security experts, existing methods which manually simulate attack steps and collect data to obtain samples, struggle to achieve automation of the attack process. As a result, they cannot ensure the efficiency of the attack simulation process.

To address C1, we introduce the concept of atomic attack behaviors, which represent the minimal units of attack activities, define their corresponding attributes, and elaborate on the meanings, contents, or calculation methods of these attributes. Subsequently, leveraging LLM, we construct a comprehensive library of atomic attack behaviors by integrating information from multiple sources, including threat actor knowledge bases, attack simulation frameworks, and penetration testing frameworks. Moreover, we infer the dependencies between these atomic attack behaviors.

To address C2, we organize attack behaviors before simulation. A prompting technique extracts TTP information from CTI reports and builds a database of known attack-technique chains; for new attack tasks, candidate technique chains are generated by reference to this database. A controlled-prompting technique for authorized research environments then asks the LLM to instantiate attack commands; commands are executed, errors fed back, and code corrected until execution succeeds. Underlying logs are collected and compressed into provenance-graph samples, closing the automation loop.

Our main contributions are summarized as follows:

- We construct a multi-source atomic attack behavior library that organizes behavior descriptions,

prerequisites, execution commands, and environment dependencies in a unified data structure.

- We propose a multi-stage TTP extraction pipeline that converts unstructured CTI reports into structured attack-technique sequences with reduced hallucination.
- We design a cost- and novelty-aware attack-chain organization method that improves the executability and diversity of candidate attack sequences.
- We build a closed-loop framework integrating attack execution, feedback-based correction, and provenance sample collection in controlled, authorized environments, providing data construction support for provenance-graph-based APT detection research.
- We demonstrate that the generated provenance-graph samples are not merely executable attack traces but can stress-test existing detection models. Experiments on four representative detectors (SLEUTH, ThreaTrace, ATLAS, Kairos) reveal clear performance degradation under our samples, making the dataset itself a contribution for evaluating detector robustness.

APTMimic differs from recent LLM-assisted offensive-security frameworks such as PentestGPT and AutoAttacker along three dimensions: (i) it targets provenance-graph-level APT samples and pairs execution with system-log auditing and lossless log compression, rather than penetration-test traces; (ii) it grounds chain planning in a domain knowledge base assembled from ATT&CK, Atomic Red Team, Metasploit, and Red Team Automation, instead of relying solely on the LLM's parametric knowledge; and (iii) chain organization is driven by an explicit cost-and-novelty objective so that the generated samples are both executable and diverse enough to challenge downstream detectors.

2. Background Knowledge

2.1. Large Language Model

LLMs are deep neural-network models built on the Transformer architecture and pre-trained on large text corpora through self-supervised learning. Their multi-billion parameter scale gives them strong semantic understanding and knowledge representation. In cybersecurity [19], LLMs are increasingly applied to threat-intelligence parsing, vulnerability analysis, phishing detection, malware analysis, and security policy generation, providing a foundation for the automated attack-simulation work that follows.

2.2. Provenance Graph-Level APT Attack Samples

A provenance graph for cyberattacks is a heterogeneous, time-stamped, directed acyclic graph in which nodes represent system entities (processes, sockets, files, etc.) and edges represent system events (reads, writes, sends, receives, deletes). Formally, P = (V, E), where each edge is a triple (subject, operation, object). The format has become the canonical data structure for attack detection, attribution, assessment, and warning, because it intuitively exposes entry points, propagation paths, and the entities involved in an attack activity.

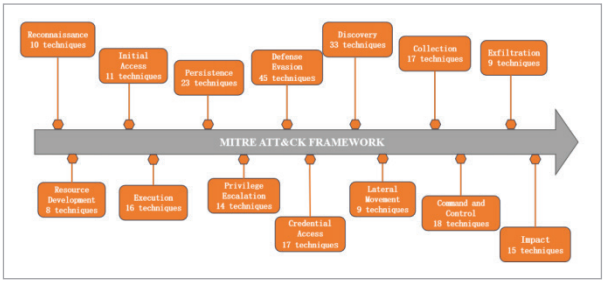
2.3. ATT&CK Model

The Adversarial Tactics, Techniques, and Common Knowledge (ATT&CK) model is a MITRE-developed cybersecurity framework [32] that systematically describes attacker tactics and techniques based on real-world attack behaviors. It covers the full attack lifecycle from initial access through data exfiltration, and currently includes 14 tactics and 236 techniques, as illustrated in Figure 1.

3. Related Work

APT attack simulation sample generation methods aim to simulate the steps and behaviors of real-world cyberattacks against target systems, networks, or applications, and collect the log data generated during

Figure 1
The framework of ATT&CK model.



the attack simulation process to construct samples. Based on different core ideas, we categorize the attack simulation methods into traditional-based attack simulation and LLM-based attack simulation.

3.1. Traditional-based Attack Simulation

Traditional-based attack simulations can be categorized into vulnerability assessment methods, penetration testing methods, and red team simulation methods.

Vulnerability-assessment methods scan target systems for weaknesses. VulMPFF [6] extracts code features from sequence, lexical-syntactic, and graph-structure perspectives, while Yamaguchi et al. [39] use code-property graphs to represent code features and discover software vulnerabilities. Such studies focus on discovery, however, and do not address how the vulnerabilities are exploited in real attack scenarios.

Penetration-testing methods evaluate system security by emulating real attackers. Hu et al. [23] generate network-topology attack trees and use deep reinforcement learning to identify optimal attack paths; Schwartz et al. [34] model penetration testing as a Markov decision process and learn optimal paths via reinforcement learning. These methods, however, focus on specific known targets or vulnerabilities and lack systematic coverage of attack techniques.

Red-team simulation methods stage end-to-end attack scenarios to evaluate overall defensive capability. DARPA's intrusion-detection evaluations [27] simulated diverse network traffic and attack patterns to construct the DARPA benchmark; Applebaum et al. [4] built a planning-based red-team emulation framework for actions after boundary penetration. Traditional red-team simulation, however, depends heavily on expert effort and is hard to automate.

3.2. LLM-Based Attack Simulation

LLMs are capable of understanding natural language, programming languages, and parsing code structures, and have strong logical reasoning and task planning abilities, all thanks to their large-scale training data. As a result, many recent studies have explored their capabilities in attack simulation tasks such as penetration testing and red team testing.

LLM-based penetration testing. LLMs integrate diverse knowledge to plan attack steps. Happe et al. [21] decompose pentesting into high-level strategy planning and low-level execution with feedback, but do not handle hallucination, which limits stability. PentestGPT [13] introduces reasoning, generation, and parsing modules, but relies entirely on the LLM's parametric knowledge without external domain grounding, limiting precise executable-code generation.

LLM-based red teaming. LLMs can construct large-scale, high-quality attack plans. AutoAttacker [38] orchestrates summarizer, planner, navigator, and experience-manager modules to automate multi-stage cyberattacks; ChainReactor [11] focuses on automated privilege-escalation chain discovery via an information extractor and AI planner. These works, however, still cover only a small fraction of the ATT&CK technique space.

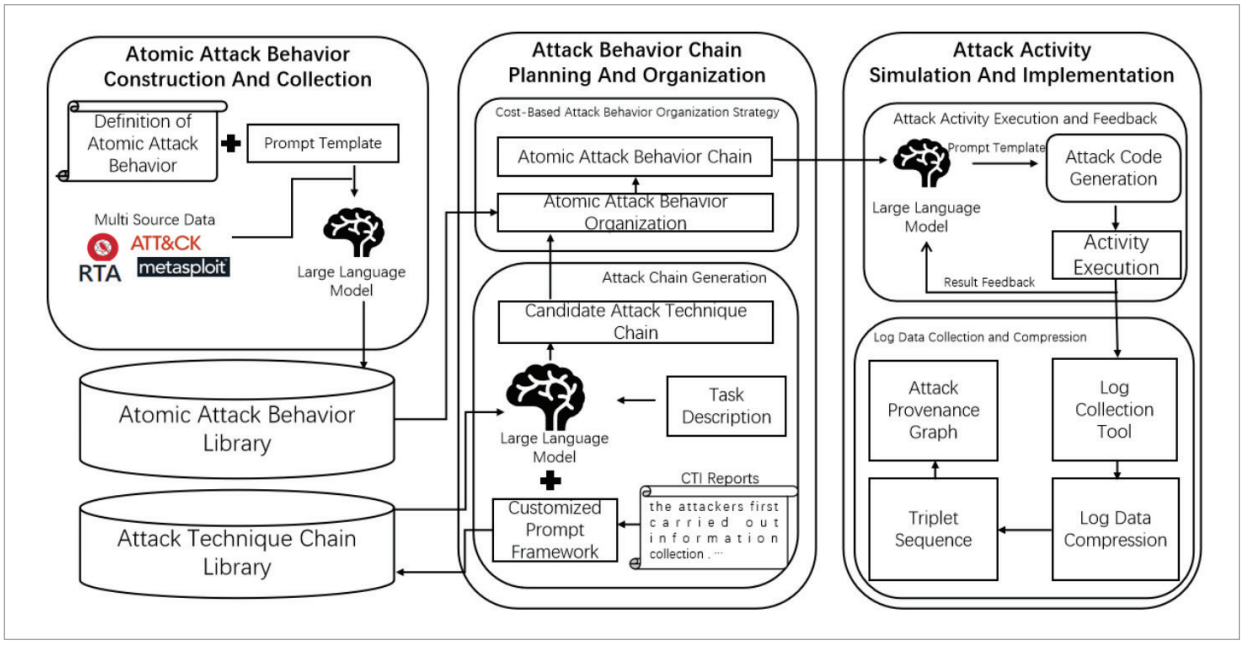
4. System Design

To enrich APT attack samples, we have designed and implemented a full-process automated attack simulation system called APTMimic, which supplements APT attack samples at the provenance graph level starting from dynamic attack simulation. This system defines and collects atomic attack behaviors that represent the minimal cost of attack activities to construct a "toolbox" usable for attack simulation processes.

Moreover, it proposes a prompt-optimized attack-technique chain planning strategy, which calculates the most novel attack-technique chains and plans the corresponding combinations of atomic attack behaviors based on cost calculations. Finally, the system obtains the core log information of the attack activities through attack code generation and system log collection and compression, and converts it into provenance-graph samples.

Figure 2 shows the overall architecture of APTMimic, comprising three subsystems: atomic-behavior collection and construction, attack-behavior chain planning and organization, and attack-activity simulation and implementation. We first define and collect atomic attack behaviors representing minimal units of attack activities and infer their dependencies. We then construct attack-technique chains via a customized prompt framework and select minimum-cost com-

Figure 2
The process of automated attack sample generation based on dynamic attack simulation.



binations of atomic behaviors using the most novel attack-technique chain as reference. Finally, we generate executable attack code, collect and compress system logs, and produce provenance-graph samples.

4.1. Atomic Attack Behavior Collection and Construction

Initially, the concept of atomic attack behavior, which represents the smallest unit of an attack activity, and its corresponding attributes are defined and explained. Subsequently, a collection strategy for atomic attack behaviors that integrates multiple information sources is designed. This strategy employs an LLM to extract atomic attack behaviors from natural language texts and command lines, and deduces the causal relationships between attack behaviors.

4.1.1. Definition of Atomic Attack Behavior:

The ATT&CK framework describes attack behaviors at the level of tactics and techniques. Although techniques represent the smallest specific attack methods, each technique can be realized through different atomic behaviors under different operating systems and implementation methods. We therefore define atomic attack behavior as the specific behavioral implementation of a particular attack technique. For example, the technique T1222.001 (File and Directory Permissions Modification) can be implemented through command-line tools such as takeown, cacls, attrib, and SubInAcl.

An attack technique can have multiple atomic attack behaviors, that is, different attack implementation forms.

To further standardize and comprehensively describe the atomic attack behaviors in the cyberattack activity space, we define eight attributes of atomic attack behaviors, their corresponding symbols, and detailed textual descriptions, as shown in Table 1. It should be noted that for the two attributes of preconditions and post-effects, we further define the common information fields they both contain as follows:

– **Execution Environment.** Refers to the operating system, OS version, virtual environment dependencies, software list, and other information required for the atomic attack behavior to run. For example, an attack exploiting the Windows EternalBlue vulnerability requires Windows 7, Windows Server 2012, or other versions that have not installed the corresponding security patches, and also needs a runtime environment such as Python and Metasploit.

Table 1
Definition of Atomic Attack Behavior Attributes.

attribute	symbol	description
ID	I	The unique identifier assigned to the atomic attack behavior
Name	N	Summarize the key points of this atomic attack behavior concisely and accurately
Description	D	Explain the purpose and detailed procedures of this atomic attack behavior
Execution Tool	T	Software and hardware tools required to execute this atomic attack behavior
Source of Information	S	Original source of information for constructing this atomic attack behavior
Attack Technique	T	Corresponding ATT&CK technique ID and name
Prerequisite	P	Prerequisites for executing this atomic attack behavior
Post-effect	F	Potential impacts of this atomic attack behavior on systems, networks, or data
Cost of behavior	C	Cost of executing this atomic attack behavior

- **Permission Information.** Refers to the user permissions and service permissions required to execute the atomic attack behavior. User permissions denote the level of authority that the user executing the attack behavior has within the system, such as regular user, administrator, etc. Service permissions refer to the access and control permissions for various services within the system.
- **File Information.** Refers to the external files, system files, credential files, and other information needed to execute the attack behavior on the target host. For example, a specific atomic attack behavior for privilege escalation on a Windows system may require the use of a Windows privilege escalation driver file, such as capcom.sys.

With regard to the behavior cost attribute, we consider it as a measure of the degree of change required to transition from the preconditions to the postconditions when executing the atomic attack behavior. The greater the degree of change, the higher the behavior cost. Therefore, we first employ the Word2Vec [29] method to obtain the embedding vectors for the precondition and the post-effect, and then calculate the cosine similarity between them. The formula for calculating the behavior cost value is as follows:

$$C(P,E) = 1 - \frac{\sum_{i=1}^{n} p_i e_i}{\sqrt{\sum_{i=1}^{n} p_i^2}\sqrt{\sum_{i=1}^{n} e_i^2}}. \quad (1)$$

The rationale behind Equation (1) is that the cost C(P, E) should approximate the extent of system-state change required for an atomic behavior to take effect. Behaviors whose post-effects differ more from their preconditions may involve broader system changes, more observable side effects, and a greater risk of execution failure. We therefore use one minus the cosine similarity between embedded precondition and post-effect descriptions as a

Figure 3
The instance of atomic attack behavior.

ID: 119
Name: Remote System Discovery
Description: Using cmd.exe on a Windows system to execute a for loop command to perform a ping scan on a specified network and output the scan results.
Execution Tool: cmd.exe
Source of Information: Atomic Red Team
Attack Technique: T1018
Prerequisite:
Execution Environment: Windows system
Permission Information: Ordinary user permissions to execute cmd.exe
File Information: Information file of the subnet to be scanned
Post-execution Impact:
Execution Environment: Windows system
Permission Information: Ordinary user permissions to execute cmd.exe
File Information: Results file of remote system discovery
Behavior Cost: 0.377

simple text-embedding-based proxy for behavioral impact; its additive form allows the downstream chain-organization algorithm to compare the total estimated costs of candidate chains.

To better illustrate the meanings of the attributes of atomic attack behaviors, Figure 3 provides a specific example of an atomic attack behavior. This example represents the atomic attack behavior of using the ping command in a Windows host to discover active host systems, corresponding to attack technique T1018.

4.1.2. Multi-Source Atomic Attack Behavior Collection

While ATT&CK provides textual descriptions for each technique, it dissects attack behaviors at the high-level text level with only a limited number of examples. Red-team organizations, hacker groups, and attack-simulation frameworks provide specific implementations that can serve as multi-source data to enrich the atomic-behavior space. We primarily collect from the following four data sources:

ATT&CK Official Website [32]. The MITRE cyber-threat-behavior knowledge base, providing definitions, purposes, use cases, defenses, and real-world examples for each tactic and technique.

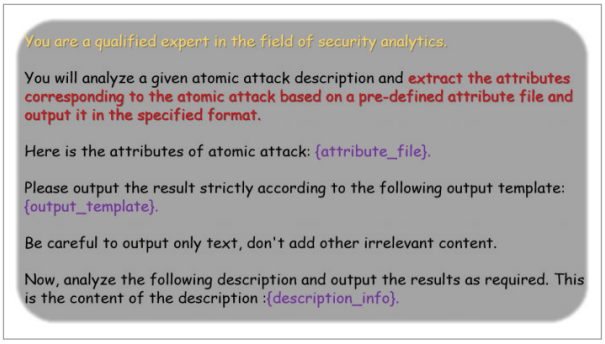
Atomic Red Team Framework [5]. An open-source attack-simulation framework that provides simple, independent, executable test cases for each attack technique.

Metasploit Framework [33]. An open-source penetration-testing framework with exploit modules across operating systems, applications, and network services, providing code implementations and operational steps.

Red Team Automation Project [25]. An open-source FireEye project providing scripts and tools to simulate common attack techniques.

We design the prompt template in Figure 4 to extract key attribute information from the four data sources and build a standardized atomic-behavior library. For data without ATT&CK technique-ID labels (e.g., Metasploit, Red Team Automation), we fine-tune the LLM on labelled technique descriptions to improve technique identification on unlabelled inputs. For potential duplicates, we use ATT&CK technique IDs as anchors: entries sharing IDs are compared on core attributes and the more credible entry is retained; entries without IDs are matched by hash values computed over core attributes; partial duplicates are merged with field complementation. Deduplication accuracy is verified through manual sampling.

Figure 4
The prompt template to construct atomic attack behavior.



4.2. Attack Behavior Chain Planning and Organization

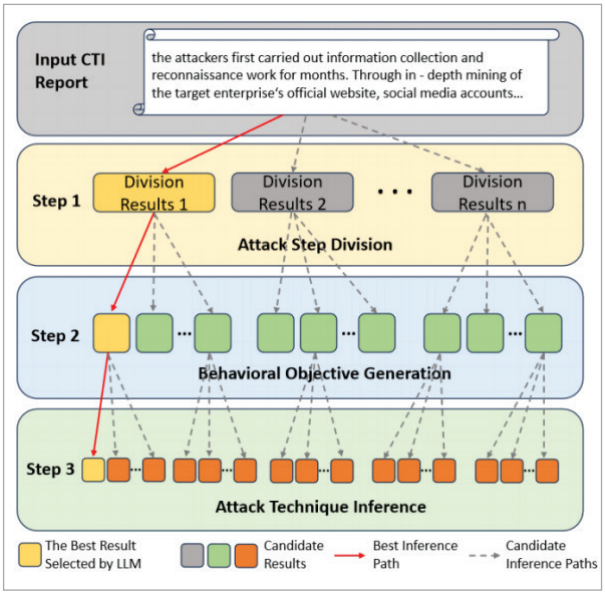
After collecting atomic behaviors, further organization is required to achieve specific attack tasks. In the candidate-chain generation phase, a new prompt framework extracts TTP information from CTI reports and builds an attack-technique chain library; given a new attack task, the LLM references this library to construct candidate chains. In the cost-based organization phase, an atomic-behavior organization algorithm uses depth-first search with pruning to obtain the chain with minimum total cost at relatively small overhead.

4.2.1. Candidate Attack Chain Generation

CTI reports, which describe the processes and technical details of real-world cyberattack activities, provide a wealth of existing attack knowledge. The attack-technique chains contained within them (i.e., TTP sequences) can guide LLM in planning simulated attack behaviors. Therefore, extracting TTP sequences of existing attack patterns from a vast number of CTI reports to construct an attack-technique chain library is of great significance.

Despite the strong zero-shot capabilities of LLMs, traditional prompting for TTP-sequence extraction from CTI reports still suffers from hallucination. To mitigate this, we propose a customized prompt-framework-based extraction solution. As shown in Figure 5, the framework has three layers: it first segments attack steps, then generates the behavioral objective of each step, and finally infers the attack

Figure 5
The process of attack-technique chain inference.



Algorithm 1 TTP sequence extraction algorithm

Require: Language model L, CTI report C, attack step planning prompt PS, attack objective identification prompt PO, attack technique extraction prompt PT, best answer selection prompt PB

Ensure: TTP sequence T

```
Define n to number of answers to be generated in each step
Define A = [] to store all answers generated
for i = 1 to n do
    Si = L(C,PS)
    for j = 1 to n do
        Oj = L(Si,PO)
        for k = 1 to n do
            Tk = L(Oj,PT)
            A += [Si,Oj,Tk]
        end for
    end for
end for
T = L(A,PB)
return T
```

techniques used in each step based on the summarized objectives. At each layer the LLM produces candidate answers and selects the best one based on report semantics, step coherence, and TTP-objective matching. Algorithm 1 details the procedure.

1. **Attack Step Segmentation:** By inputting customized prompts P_S and the complete CTI report, the LLM is instructed to segment the entire CTI report into different parts according to attack steps, generating n candidate segmentation results S_i.
2. **Behavioral Objective Generation:** By inputting customized prompts P_O and a particular attack step segmentation result S_i from the previous step, the LLM is prompted to infer the attack behavioral objective of each step based on its description, generating n candidate behavioral objective results O_j.
3. **Attack Technique Inference:** By inputting customized prompts P_T and a specific attack behavioral objective O_j from the previous step, the LLM is instructed to infer the TTP sequence it represents based on the detailed description of each attack step and the corresponding behavioral objective, generating n candidate TTP sequence results T_K .
4. **Result Generation:** By inputting customized prompts P_B and all TTP sequence inference chains $[S_i,O_j,T_k]$, the LLM is instructed to select the best TTP sequence result Tand output it.

After obtaining a large number of CTI reports and the attack-technique chains they contain, the LLM can use this as empirical data to help plan attack-technique chains that can achieve attack goals for new attack simulation tasks. Specifically, given the initial state of an attack simulation task and the final attack objective, the LLM will leverage the Retrieval-Augmented Generation (RAG) technique to quickly filter out relevant information snippets from the empirical database through similarity calculations and infer credible candidate attack-technique chains.

To favor diversity, we propose a novelty measurement for candidate attack-technique chains that references the chain library. We define a pair-wise frequency score f (fraction of library chains containing both techniques Ti and Tj) and a distance score D (average positional distance of Ti and Tj across chains in which both appear). The local novelty contribution of an adjacent technique pair combines f and D; the chain-level score is the sum over all adjacent pairs. A higher score means a rarer combination with greater inter-technique span. Equations (2)-(6) give the formulas.

$$f(i,j)=\frac{\sum_{t=1}^{R} IsIn(T_i,L_t)\cdot IsIn(T_j,L_t)}{R},\ i,j\in[1,K] \quad (2)$$

$$D(i,j)=\frac{\sum_{t=1}^{R} Dis(T_i,T_j,L_t)}{\sum_{t=1}^{R} IsIn(T_i,L_t)\cdot IsIn(T_j,L_t)},\ i,j\in[1,K] \quad (3)$$

$$IsIn(T_i,L_t)=\begin{cases}1 & if\ T_i\ in\ L_t\\ 0 & otherwise\end{cases} \quad (4)$$

$$Dis(T_i,T_j,L_t)=\begin{cases}\frac{|Ind(T_i)-Ind(T_j)|}{|L_t|} & if\ T_i\ and\ both\ in\ L_t\\ 0 & otherwise\end{cases} \quad (5)$$

$$Novelty(C_N)=\sum_{i=1}^{N-1}\frac{1-f(b_i,b_{i+1})+D(b_i,b_{i+1})}{2}. \quad (6)$$

Equation (6) is defined this way because the frequency component (1 - f) captures how rare the co-occurrence of two techniques is across known chains, while the distance component D captures how atypical their ordering is. Averaging the two per adjacent pair and summing along the chain therefore prefers chains that are simultaneously rare and structurally non-standard, which is the bias needed for generating samples that stress-test existing detectors.

4.2.2. Cost-Based Attack Behavior Organization Strategy

The selection of the attack behavior organization strategy is based on the most novel attack-technique chain generated by the LLM as a reference standard. According to this most novel attack-technique chain, and referring to the previously constructed library of atomic attack behaviors, the high-level, macroscopic attack techniques are replaced with specific, practical attack behaviors to obtain concrete behavioral implementation methods that can achieve the attack goal.

For all atomic attack behaviors with known behavior costs under each attack technique, in order to obtain the combination strategy with the minimum behavior cost and avoid the huge overhead produced by brute-force search and concatenation of atomic attack behaviors, we implement an atomic attack behavior organization algorithm using the depth-first search algorithm combined with the proposed pruning strategy, as shown in Algorithm 2, which includes the following five steps:

Algorithm 2 Atomic attack behavior combination algorithm

Require: LLM-generated latest attack-technique chain CN = [T1,T2,...,Tn], the set of atomic attack behaviors Ai = [a1,...,aj] corresponding to each technique Ti in CN

Ensure: Attack behavior organization strategy Paction

```
Define N to length of CN
Define Bestcost to the cost of current optimal attack behavior combination
Paction = []
ActionCombination (K,Pcurrent,Currentcost)
    /* K represents current loop index */
    /* Currentcost represents the whole cost until current loop */
    /* Pcurrent = [] represents the list to store attack behavior combination until current loop */
if K == N and Currentcost < Bestcost then
    Paction = Pcurrent
    Bestcost = Currentcost
    return
end if
if Current_cost + sum_{i=K}^{N} min{a_i.C | a_i ∈ A_i} > Best_cost then
    return
end if
for aj in AK do
if Pcurrent[-1].E → aj.P then
ActionCombination(K +1,Pcurrent.append(aj), Currentcost + aj.C)
    end if
end for
ActionCombination(0,null,0)
return Paction
```

1 **Function Definition:** A function (K , ,) is defined based on the existing attack-technique chain and the set of atomic attack behaviors corresponding to each technique , to organize the atomic attack-behavior chain . Here, represents the current recursive round (i.e., the atomic attack behavior corresponding to the K-th attack technique being constructed in); represents the atomic attack-behavior chain constructed before the current recursive round; represents the cumulative behavior cost before the current recursive round. $ActionCombination P_{current} Current_{cost} C_N A_i T_i P_{action} KC_{N-} P_{current} Current_{cost}$

2 **Behavior Chain Update Strategy:** When $K = N$, it represents that a viable atomic attack-behavior chain has been constructed. If the total cost of this behavior chain is less than the existing minimum cost, then update the current minimum cost $Best_{cost}$ and the optimal attack-behavior chain P_{action}, concluding the current recursive round.

3 **Pruning Strategy:** If the cumulative behavior cost $Current_{cost}$ before the current recursive round and the sum of the minimum atomic attack behavior costs corresponding to the subsequent K to N attack techniques are greater than $Best_{cost}$, it implies that even if the subsequent selected atomic attack behaviors all have the smallest behavior costs, it is impossible to obtain an optimal solution. Therefore, there is no need for further operations, and the current round of recursion is terminated.

4 **Behavior Chain Combination Strategy:** If neither of the previous two strategies applies, then start selecting the atomic attack behaviors corresponding to the K-th attack technique. Specifically, iterate through the available attack behaviors a_i from A_K. If the postcondition of the previously selected atomic attack behavior $P_{current}[-1]$ can deduce the prerequisite condition P for a_i, then concatenate the attack behavior a_i to the existing solution $P_{current}$, and update the current cost $Current_{cost}$. Then, enter the next level of recursion $ActionCombination(K+1, P_{current}.append(a_i), Current_{cost}+a_i.C)$, and begin selecting the atomic attack behavior corresponding to the $(K+1,)$-th attack technique.

5 **Run and Output Results:** Invoke $ActionCombination(0, null, 0)$ to ultimately obtain the optimal solution P_{action} and the corresponding behavior cost $Best_{cost}$.

4.2.3. Attack Activity Simulation and Implementation

This subsystem first proposes a controlled-prompting technique designed for authorized research environments to generate executable attack commands. It then executes the commands and corrects the code based on execution feedback to obtain feasible attack implementations. Subsequently, the subsystem collects underlying logs during attack simulation and applies log-compression algorithms to construct usable provenance-graph samples.

1 Attack Activity Execution and Feedback:

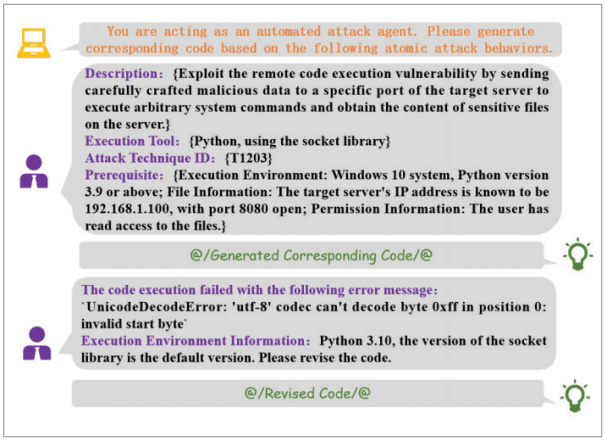
To convert the atomic-behavior chain into executable instructions, prior work [26], [15] has demonstrated the effectiveness of LLMs for automated code generation. However, LLMs apply defensive policies that refuse to generate sensitive attack code. We therefore use a domain-scoped role-prompting technique. Under explicit authorization for security research, the model is instructed to act as an automated attack agent within a controlled research scenario, and is fed the descriptions, execution tools, preconditions, and attack techniques of the atomic behaviors. The LLM then generates the necessary commands and code.

Generated code can still fail at execution time due to environment-specific constraints, contextual dependencies, or minor syntax issues. For failing atomic-behavior code we record the original code, error message, and execution-environment details and feed them back to the LLM for code correction. As Figure 6 illustrates, a UTF-8 decoding error caused by unparseable bytes in a socket response is fixed by appending errors='ignore' in the response-reception phase, which the LLM identifies and applies after seeing the feedback.

Then, the user feeds back the error details and current execution environment information to the LLM. Finally, the LLM successfully corrects the code by adding the parameter 'errors = 'ignore'' in the server response reception phase.

Figure 6

The instance of attack execution code modification.



2 Log Data Collection and Compression:

System logs record the operational status, events, and actions of operating systems or applications.

Each log entry contains a <subject, action, object> triplet, and these triplets can be linked end-to-end to form a connected provenance graph representing system activities [8]. Since APTMimic's goal is to produce provenance-graph-level APT samples, during simulation we use log-collection tooling to capture underlying logs and convert them into usable provenance-graph samples.

We use SPADE [18], a distributed system-log auditing tool, to collect system logs generated by the target host while executing the atomic-behavior chain. SPADE deploys probes at key system nodes to capture process interactions, file read/write operations, and network communications, correlating heterogeneous operational information into a contextually enriched stream. Its modular architecture supports different systems and adjustable collection granularity, with real-time processing for prompt mining and auditing.

Raw system logs contain considerable redundancy, such as semantically duplicate events or repeated entity interactions. Before extracting provenance-graph triplets, redundant-data compression is therefore required. Existing solutions either lose significant semantics or incur high computational cost. We adopt PDCleaner [24], a recent semantically lossless log-compression method that combines deletion of semantically similar redundant events, aggregation of operationally similar repeated entities, and merging of behaviorally similar overlapping event chains, all within bounded operational cost.

5. Experimental Evaluation

This subsection first employs quantitative statistical analysis to evaluate the comprehensiveness of atomic attack behavior construction; then uses precision and recall to assess the effectiveness of attack-technique chain extraction; subsequently evaluates the feasibility of the generated attack-behavior chains through success rates and interaction rounds; and finally tallies the human effort and costs saved by conducting attack simulation activities.

5.1. Experimental Setup

1 **Dataset:** We collected a total of 1,898 attack-related articles from ATT&CK, Atomic Red Team, Metasploit, and Red Team Automation to construct a comprehensive library of atomic attack behaviors. We also gathered 1,000 CTI reports from 2020 to 2024 from 30 security vendors, blogs, and forums to build the attack-technique chain library, and selected 300 reports with detailed TTP sequence results as the reference standard.

CTI corpus construction. Reports were collected from 30 widely cited security vendors and forums (e.g., CrowdStrike, FireEye/Mandiant, Kaspersky, Symantec, McAfee, Cisco Talos) published 2020-2024. A report was retained if it was English-language, described at least three identifiable ATT&CK-aligned attacker steps, and had not been retracted. From the 1,000 retained reports, 300 reports with self-contained, fully attributable attack narratives whose steps map one-to-one to ATT&CK technique IDs formed the benchmark subset. Reference TTP chains were validated by a two-round annotation procedure: two senior analysts labelled each report against ATT&CK v15 independently, with a third analyst arbitrating disagreements; only chains achieving three-way consensus were used as ground truth.

2 **Experimental Environment Configuration and Module Parameter Settings:** All experiments were conducted on a Windows host with an Intel i9-13900K CPU (24 cores, 3.00 GHz), 64 GB RAM, and an NVIDIA RTX 4090 GPU. We set up a local network using Oracle VirtualBox with two Linux hosts (Kali Linux 2023.04, Ubuntu 22.04) and two Windows hosts (Windows 10, Windows Server 2016), enabling lateral movement, command-and-control communication, and data leakage. Basic software components related to typical APT vectors were installed on each machine. GPT-4o was used as the primary LLM.

No public cloud platform or local cloud emulator was used. The Web Storage Service Exploitation task was implemented as a locally hosted web storage application inside the isolated VirtualBox network; it evaluated exploitation of a vulnerable web-facing service and subsequent access to files stored on that host.

All attack simulations reported here were carried out exclusively in an isolated, physically air-gapped laboratory network owned by the authors' institution; no external system or production environment was contacted at any point. Hosts were

dedicated and re-imaged after each run, and the work was conducted under the institution's authorized security-research procedure. Following responsible-disclosure principles, released artifacts include only benchmark labels, evaluation scripts, and high-level chain descriptions; concrete exploit code, fine-tuned prompts that trigger sensitive command generation, and full provenance-graph payloads are withheld and shared only with verifiable academic groups under data-use agreement.

To ensure that the attack-technique chain planning strategy achieves high precision and limited cost expenditure when extracting TTP sequences, we set the number of candidate answers generated per round to 3. All the above parameters were determined through grid search.

We probed alternative values via grid search on a 50-report development split. Using 1 candidate dropped TTP-extraction precision by 9.6 points and recall by 7.4 points because the LLM had no opportunity to reject its own hallucinated answer; using 5 candidates yielded only a 0.4-point precision improvement while almost doubling token cost. Three candidates therefore sit at the knee of the cost-quality curve.

5.2. Coverage of Atomic Attack Behavior Construction

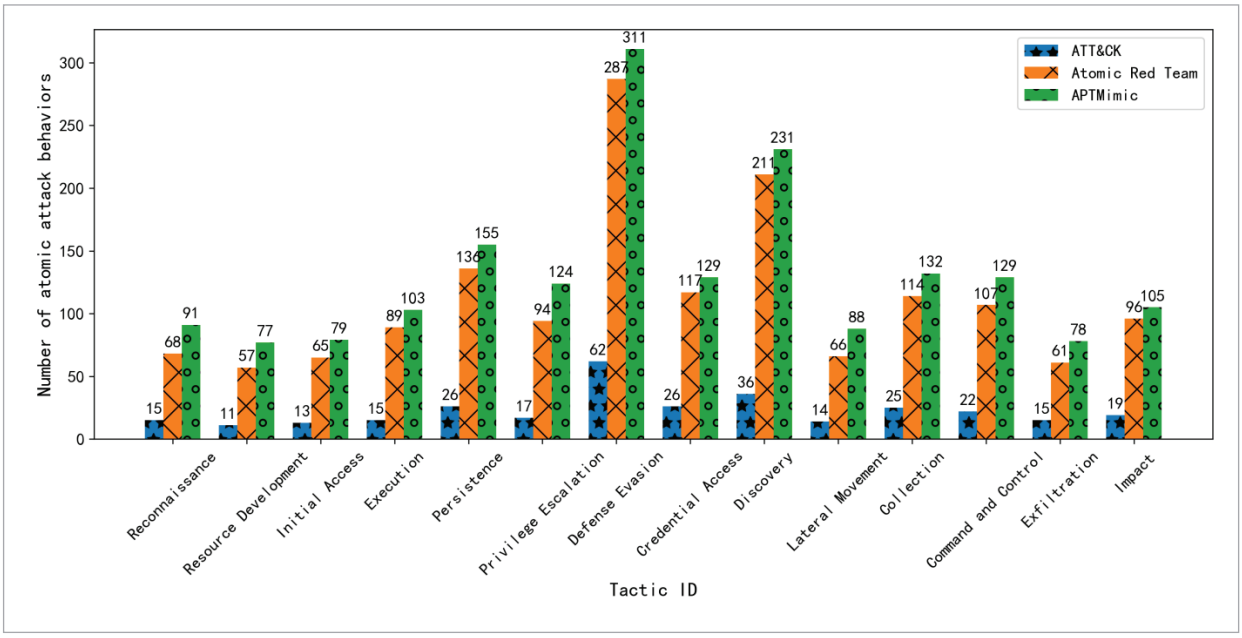
This subsection evaluates whether our atomic-behavior library expands existing coverage. In Submodule 1, this work built a library of atomic attack behaviors that can implement a specific attack technique based on multiple information sources.

This library encompasses a broad space of attack behaviors. To demonstrate the comprehensiveness of this database, we compare the number of atomic attack behaviors covered by it with those covered by the ATT&CK model and the Atomic Red Team project. The ATT&CK model provides the standard definitions of tactics, techniques, and procedures (TTPs) for cyberattack behaviors, while Atomic Red Team is a popular red team testing toolkit that offers specific implementations for each attack TTP.

Figure 7 reports the coverage of atomic attack behaviors across fourteen attack tactics. In this experiment, the APTMimic library contains 480% more atomic-behavior entries than the ATT&CK technique baseline and 16.8% more than the Atomic Red Team collection. ATT&CK primarily standardizes tactics and techniques and provides selected procedural examples, whereas Atomic Red Team supplies executable tests for many techniques. The

Figure 7

The coverage of atomic attack behaviors of APTMimic for different attack tactics.



additional entries in APTMimic mainly arise from integrating these sources with Metasploit and Red Team Automation and from organizing behaviors for proprietary protocols, business-logic vulnerabilities, and complex multi-stage intrusion scenarios. These results demonstrate broader coverage within the sources and counting criteria used in this study, rather than universal superiority over all existing behavior libraries.

5.3. TTP Extraction Accuracy

This subsection evaluates the proposed CTI-driven TTP-sequence extraction method against three baselines: the classic TTPDrill, the state-of-the-art AttacKG, and an LLM model with Chain-of-Thought (CoT) prompting [37]. The CoT baseline is included to show that the gain is not solely due to LLM reasoning but also to the prompt strategy we design. We selected six security vendors with diverse narrative styles - CrowdStrike (behavior-focused), FireEye (motive-focused), Kaspersky (corporate-incident-focused), Symantec (technology-innovation-focused), McAfee (data/visualization-focused), and Cisco (defense-strategy-focused) – and collected 50 TTP-describing reports per vendor as the test benchmark.

Table 2 shows the precision and recall of TTP-sequence extraction across the six vendors. The customized prompt framework attains the best results on all vendors, with an average precision of 0.938 and an average recall of 0.943. TTPDrill, which uses semantic-similarity matching against attack-technique ontologies, identifies techniques through a limited set of verbs and is hard to control for precision; its mechanical sentence segmentation also matches each sentence against techniques without context, yielding many matches and a recall that exceeds precision by a large margin.

Why some CTI sources are easier than others: vendors using step-by-step attacker narratives (CrowdStrike, Cisco) yield reports where steps map almost linearly to ATT&CK techniques, favoring all methods; vendors embedding techniques in motive analysis or strategic commentary (FireEye, Kaspersky) expose the limits of TTPDrill's verb matching and AttacKG's static subgraph matching, which is where our framework gains the most. Methodologically, this confirms that the gain comes from semantically anchoring techniques rather than from raw LLM capacity, validating the three-layer prompt design.

AttacKG converts TTP-sequence extraction into a graph-pattern matching problem: it models the report as a graph and constructs subgraphs representing attack techniques, then matches the technique subgraphs into the report graph. Compared with TTPDrill, APTMimic considers behavioral interactions between system entities and builds a more comprehensive subgraph ontology, hence its precision and recall are higher. However, AttacKG relies entirely on graph matching and cannot capture complex semantics or context, leaving a substantial gap relative to LLM-based methods.

Comparing the two LLM-based extractors, our prompt framework better leverages LLM capabilities. When CTI sentences are semantically unclear or too brief, the LLM tends to hallucinate or over-re-

Table 2
The Result of Attack Technique Chain Extraction.

Security Vendor	TTPDrill		AttacKG		COT		APTMimic	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
CrowdStrike	0.292	0.573	0.787	0.673	0.898	0.908	0.933	0.963
FireEye	0.182	0.665	0.679	0.595	0.864	0.855	0.919	0.964
Kaspersky	0.221	0.703	0.627	0.557	0.823	0.853	0.955	0.962
Symantec	0.354	0.563	0.832	0.702	0.892	0.892	0.991	0.974
McAfee	0.292	0.494	0.694	0.584	0.835	0.834	0.925	0.903
Cisco	0.177	0.557	0.571	0.543	0.795	0.825	0.905	0.892

ly on keywords; with our framework, candidate-answer analysis suppresses hallucination and converges to a better answer.

5.4. Execution Feasibility of Generated Attack Behavior Chains

This subsection evaluates whether the generated attack-behavior chains succeed on their objectives. In Submodule 2, we converted the most novel attack-technique chain into the minimum-cost atomic-behavior chain. In Submodule 3, we generated implementation code for each atomic behavior and ran it. After discussions with security experts, we designed 12 tasks covering different attack stages and scenarios (see Table 3). To prevent accidental results, each task was repeated three times and all results recorded. We further compared APTMimic with AutoAttacker and PentestGPT, both reimplemented with the same GPT-4o model as APTMimic and temperature 0.5.

The 12 tasks cover early-stage operations such as file write, privilege escalation to obtain higher privileges, lateral-movement and control-expansion behaviors (credential theft, hash dumping, pass-the-hash), service-specific attacks (MySQL hash dumping), vulnerability exploitation (Log4j), and persistent access (SSH-public-key injection), spanning all critical stages from initial access to multi-stage intrusion, lateral expansion, and target control.

For fair comparison, the three systems used the same GPT-4o version with temperature 0.5, top-p 0.95, and identical maximum context length; all three received the same atomic-behavior descriptions and the same task brief, while PentestGPT retained its native vulnerability-knowledge module and AutoAttacker its experience manager, since stripping these would not be a faithful reproduction; all three ran on the same isolated VirtualBox network; and the only baseline modifications were the I/O wrappers needed to drive them on our benchmark.

Table 3
The List of Attack Simulation Activities.

Attack Activity	Description	ATT&CK Technique Chain
File Write	Implant a file in a specific user's directory on the target system	T1105
Privilege Escalation	Escalate from a regular user to root in Linux	T1068
Credential Theft	Dump user NTLM hashes from LSASS memory on the target system	T1003.001
File Upload	Exploit a public-facing file-upload vulnerability and deploy a web shell	T1190, T1505.003
Script Execution	Exploit command injection in a Linux service and execute a malicious shell script	T1190, T1059.004
Hash Dumping	Dump local user password hashes from the Windows SAM database	T1003.002
Pass-the-Hash	Obtain an NTLM hash and use it over Windows administrative services to access another server	T1003.001, T1550.002, T1021.002
MySQL Hash Dump	Extract MySQL account hashes, crack a password, and authenticate to the database	T1213.006, T1110.002, T1078
Log4j Exploit	Exploit a public-facing Log4j-vulnerable service and execute commands on the Linux host	T1190, T1059.004
SSH Public Key Injection	Add an attacker-controlled key to the target account's SSH authorized_keys file	T1098.004
Advanced Lateral Movement	Exploit an initial public-facing vulnerability, spray passwords, and use valid accounts for remote access	T1190, T1110.003, T1078, T1021.004
Web Storage Service Exploitation	Exploit a vulnerable locally hosted web storage service and access files stored on the host	T1190, T1005

Table 4
The Result of Attack Behavior Chain Execution.

Attack Activity	AUTOATTACKER		PENTESTGPT		APTMimic	
	Result	Round of Interaction	Result	Round of Interaction	Result	Round of Interaction
File Write	3/3	3.0	3/3	4.3	3/3	4.0
Privilege Escalation	3/3	3.3	3/3	4.0	3/3	4.0
Credential Theft	3/3	3.7	3/3	4.3	3/3	4.3
File Upload	2/3	3.0	3/3	5.0	3/3	4.7
Script Execution	3/3	4.0	3/3	5.7	3/3	5.0
Hash Dumping	2/3	4.3	2/3	5.7	3/3	5.3
Pass-the-Hash	1/3	4.3	3/3	5.3	3/3	5.7
MySQL Hash Dump	2/3	5.0	2/3	7.0	3/3	6.3
Log4j Exploit	1/3	4.7	2/3	5.3	3/3	6.0
SSH Public Key Injection	2/3	5.0	2/3	7.3	3/3	6.7
Advanced Lateral Movement	0/3	7.3	2/3	8.0	2/3	8.0
Web Storage Service Exploitation	1/3	7.3	1/3	8.7	2/3	8.0

ATT&CK mapping rationale. The mappings follow the behavior definitions in MITRE ATT&CK: T1105 covers transferring a file or tool into a compromised environment; T1068 covers exploitation for privilege escalation; T1003.001 and T1003.002 cover credential extraction from LSASS memory and the SAM database, respectively; T1190 covers exploitation of a public-facing application; T1505.003 covers web-shell persistence; T1059.004 covers Unix-shell command execution; T1550.002 covers pass-the-hash authentication and T1021.002 covers SMB/Windows administrative services; T1213.006 covers data obtained from databases, T1110.002 password cracking, and T1078 use of valid accounts; T1098.004 covers modification of SSH authorized keys; T1110.003 covers password spraying and T1021.004 SSH-based remote services; and T1005 covers collection of data from a local system. Technique order follows the operational sequence described for each task.

The execution results of APTMimic, AUTOATTACKER, and PENTESTGPT in the 12 different attack tasks, as well as the average number of interactions with the LLM, are shown in Table 4. Analyzing the average success rate of the attack activities, excluding the two task failures due to command-line parameter setting errors in complex attack environments, APTMimic succeeded in 34 out of 36 attack simulations, achieving a success rate as high as 94.4%, which is significantly higher than the 63.9% of AutoAttacker and 80.6% of PentestGPT. The reason is twofold: AutoAttacker maintains an experience manager but lacks broad external attack knowledge, leading to weak performance on complex tasks; PentestGPT separates reasoning, generation, and parsing modules but focuses on vulnerability identification and code analysis, limiting it on complex multi-stage tasks. APTMimic deconstructs complex attacks at the atomic-behavior level and grounds chain planning in a comprehensive behavior library, which yields the higher success rate.

Run-to-run stability and failure cases. With each task repeated three times, APTMimic produced an identical 3/3 outcome for 10 of 12 tasks; Advanced Lateral Movement and Web Storage Service Exploitation each yielded 2/3. Interaction-round counts varied within +/-1 across runs, indicating largely stable behavior. The two failed runs were concentrated in environment-sensitive parameter binding: the lateral-movement payload assumed the default WMI

namespace while the target used a customized mapping, and the locally hosted web storage task used an application-specific endpoint parameter that did not match the deployed service configuration. These cases show that robustness depends on the freshness and completeness of environment-specific dependency information, motivating the dependency-library extension discussed in Section 6.

APTMimic averages 5.67 interaction rounds and PentestGPT 5.88, both above AutoAttacker's 4.58. APTMimic and PentestGPT split complex tasks into specialized subtasks (chain planning, cost re-organization, feedback-driven correction), each requiring separate LLM exchanges, whereas AutoAttacker uses a simpler loop with its experience manager. The additional rounds are absorbed by the cost/novelty re-organization and feedback-driven correction stages: they buy reliability, not verbosity, and trade modest inference cost for substantially higher success rate.

5.5. Challenge Evaluation of Generated Dynamic Provenance Graph Samples on Existing Attack Detection Models

This subsection evaluates whether the generated novel attack samples challenge existing detection models. Specifically, four representative attack detection models are selected as the evaluation targets, including the rule-based attack detection system SLEUTH, the node-level attack detection system THREATRACE, the edge-level attack detection system ATLAS, and the graph-level attack detection system Kairos. Precision, Recall, and F1-score are adopted as evaluation metrics to analyze the performance of each model in attack detection.

We generate 12 dynamic provenance-graph samples from the 12 attack-simulation tasks and use them as test datasets to evaluate existing detection models on previously unseen attack samples. Table 5 lists the basic information of these datasets, including node and edge counts and the number of subgraphs obtained by fixed-time-window partitioning. The generated samples contain substantial graph scale, multi-step behavior chains, and background-event noise, providing controlled stress-test data for evaluating representative detectors under the experimental conditions of our isolated laboratory environment.

Table 6 shows that for node-level detection, SLEUTH achieves only 0.620 precision and 0.511 recall: its predefined rules and label-propagation cannot characterize evolving complex attacks or cover

Table 5
The Description of Samples Generated by Attack Simulation.

Attack Activity	Nodes	Edges	Subgraphs
File Write	3654	12386	26
Privilege Escalation	4211	16841	30
Credential Theft	4892	20912	34
File Upload	2832	9473	18
Script Execution	3498	10746	23
Hash Dumping	5447	35047	34
Pass the Hash	6934	41923	45
MySQL Hash Dumping	10928	59348	51
Log4j Exploit	8124	39562	40
SSH Public Key Injection	12384	72340	50
Complex Lateral Movement	15023	85739	55
Web Storage Service Exploit	17937	94572	58

Table 6
Attack Detection System Performance Evaluation.

Test Model	Precision	Recall	F1 Score
SLEUTH	0.620	0.511	0.560
THREATRACE	0.649	0.717	0.681
ATLAS	0.601	0.679	0.637
Kairos	0.729	0.782	0.754

emerging patterns, yielding high false-positive and false-negative rates. ThreaTrace adopts semi-supervised anomaly detection on benign data and reaches 0.717 recall through sensitivity to anomalies, but the lack of attack prior knowledge and the benign/malicious class imbalance still cause many false alarms on novel attack nodes, keeping precision low.

In the edge-level detection task, ATLAS achieves 0.601 precision and 0.679 recall. Edge-level detection essentially corresponds to event-level detection, which is finer-grained and more susceptible to data sparsity and noise. In addition, ATLAS uses under-sampling and over-sampling to balance the dataset, which can weaken its ability to learn complex attack-sequence features and produces more false positives and negatives.

In the graph-level detection task, Kairos achieves 0.729 precision and 0.782 recall, outperforming the node-level and edge-level methods because graph-level detection operates at coarser granularity and is more likely to capture global anomaly patterns. Even so, Kairos still produces considerable false positives and negatives on our novel samples, as it quantifies anomaly via deviation from historical benign patterns; when malicious behaviors are embedded within normal system-behavior chains, its detection capability is significantly constrained.

In summary, all four representative detectors perform unsatisfactorily on the dynamic samples from the 12 attack-simulation tasks, indicating clear limitations of current methods on unknown and novel attack-behavior chains. Our generated samples thus pose an effective challenge to existing detectors and serve as valuable resources for assessing detector robustness and generalization. Methodologically, this finding promotes the dynamic samples from a by-product to a first-class contribution: they form a stress-testing dataset that exposes failure modes across rule-based, node-level, edge-level, and graph-level detectors alike.

5.6. Overhead Experiment

5.6.1. Human Effort Savings Assessment

This subsection measures how much human effort our automated pipeline saves. To conduct the test, this experiment recruited 10 security experts with over five years of experience on the red team. They were asked to manually perform the four-stage full-process security tasks of task planning, tool collection, code generation, and attack execution based on the 12 attack tasks designed above. The average time spent by the security experts and APTMimic in each behavioral stage was then calculated.

Table 7 shows the average time spent by security experts and APTMimic in each behavioral stage. Experienced cybersecurity experts require an average of 111.72 minutes per cyberattack activity. APTMimic, with its pre-constructed atomic-behavior library, LLM generation, and manual algorithms, dramatically reduces the search time in the first three stages, completing each attack task in an average of 6.14 minutes - a 105.58-minute saving per operation.

Table 7
The Time Cost by Human and Aptmimic.

Behavioral Phase	Manual Time Consumption	APTMimic Time Consumption	Time Saving
Task Planning	41.65min	0.53min	41.12min
Tool Collection	36.51min	0.70min	35.81min
Code Generation	22.89min	1.12min	21.77min
Attack Execution	10.67min	3.79min	6.88min

5.6.2. Cost Assessment

Table 8 reports the per-stage LLM costs for all 12 attack-simulation tasks. With the assured execution-success rate, APTMimic requires only an average of $0.54 to complete a full attack activity. We also computed the one-time cost of constructing the atomic-behavior library and the attack-technique chain library, which totals $217.3; this is a one-off expenditure, after which both knowledge bases can be reused indefinitely.

Table 8
The Cost of Attack Simulation.

Attack Activity	Task Planning	Tool Collection	Code Generation	Attack Execution
File Write	0.05 USD	0.08 USD	0.13 USD	0.02 USD
Privilege Escalation	0.04 USD	0.15 USD	0.12 USD	0.04 USD
Credential Theft	0.05 USD	0.10 USD	0.11 USD	0.05 USD
File Upload	0.08 USD	0.11 USD	0.13 USD	0.04 USD
Script Execution	0.10 USD	0.09 USD	0.13 USD	0.05 USD
Hash Dumping	0.11 USD	0.10 USD	0.18 USD	0.05 USD
Pass-the-Hash	0.13 USD	0.16 USD	0.20 USD	0.07 USD
MySQL Hash Dump	0.17 USD	0.16 USD	0.24 USD	0.10 USD
Log4j Exploit	0.18 USD	0.13 USD	0.21 USD	0.09 USD
SSH Public Key Injection	0.17 USD	0.15 USD	0.20 USD	0.14 USD
Advanced Lateral Movement	0.22 USD	0.20 USD	0.27 USD	0.24 USD
Web Storage Service Exploitation	0.25 USD	0.21 USD	0.25 USD	0.20 USD

6. Discussion and Future Work

In our work, we elaborate on the design, implementation, and evaluation process of APTMimic, a dynamic APT attack sample generation system based on LLM attack simulation. The system is capable of automatically generating attack behaviors targeted at specific attack activity objectives, while simultaneously providing corresponding provenance-graph samples.

Experimental results show that, under the sources and counting criteria used in this study, the APTMimic library contains 16.8% more atomic-behavior entries than Atomic Red Team. The proposed method achieves 0.938 precision on CTI-report TTP-sequence extraction and completes 34 of 36 attack-simulation runs (94.4%) with an average per-task time of 6.14 minutes. In the evaluated benchmark, its execution success rate is higher than those of AutoAttacker and PentestGPT. Two aspects remain for improvement.

Scope of applicability. The 94.4% success rate is bounded to a specific operating envelope: isolated laboratory network, small heterogeneous Linux/Windows topology, techniques covered by our atomic-behavior library, and CTI styles within the 2020-2024 distribution. The framework transfers well to provenance-graph dataset construction for academic detector evaluation, but less well to (i) production or actively monitored environments where defensive instrumentation can preempt generated commands; (ii) APT techniques requiring zero-day exploits or hardware-specific primitives; and (iii) low-resource or non-English CTI. Manual intervention remains useful for unknown service versions or customer-specific topologies outside the laboratory. Readers should not generalize the 94.4% beyond these constraints.

Adaptability. APTMimic succeeded in 34/36 simulations but failed twice in complex scenarios with brittle environment-dependency binding. To improve this, future work will construct a comprehensive environment-dependency library that systematically organizes target-environment configurations, enabling rapid matching and significantly improving robustness on complex tasks.

Real-time performance. LLM inference speed currently limits the timeliness of sample generation,

which may cause generated samples to lag behind the latest attack dynamics. Future work will explore knowledge distillation to transfer large-model knowledge into smaller models, maintaining performance while significantly improving inference speed for real-time APT-behavior generation and log collection.

Responsible-use posture. APTMimic is strictly a research tool for the defensive evaluation of provenance-graph-based APT detectors in authorized environments. We distinguish two artifact categories. Sharable: benchmark task list, atomic-behavior library schema, ground-truth TTP labels, detector evaluation scripts, and aggregate provenance-graph statistics. Withheld: full atomic-behavior implementations containing sensitive command sequences, fine-tuned prompts that produce them, and raw provenance graphs in non-obfuscated form. Withheld artifacts are released only to verifiable academic groups under signed data-use agreement.

7. Conclusion

This work introduces APTMimic, an automated APT-sample generation system that couples a multi-source atomic-behavior library, a customized prompt framework for CTI-driven TTP extraction, and a cost- and novelty-aware chain-organization algorithm with an authorized closed-loop simulation pipeline. APTMimic produces executable attack chains and corresponding provenance-graph samples, providing both an automation framework and a stress-testing dataset for provenance-graph-based APT detection. Experiments across 12 attack tasks show that APTMimic substantially extends existing attack-behavior coverage, achieves high TTP-extraction precision, and reliably executes complex chains in an isolated laboratory. The generated samples reveal clear performance limits in four representative detectors, demonstrating the dataset's value for downstream detector evaluation.

Acknowledgment

This work is supported in part by the following grants: Wenzhou Key Scientific and Technological Projects (No. ZG2024007), Wenzhou Basic Scientific Research Projects (No. G20240033), Key Laboratory of Data Intelligence and Governance of Wenzhou City (No. KLDI2025009).

References

1. Alshamrani, A., Myneni, S., Chowdhary, A., Huang, D. A Survey on Advanced Persistent Threats: Techniques, Solutions, Challenges, and Research Opportunities. IEEE Communications Surveys & Tutorials, 2019, 21(2), 1851-1877. https://doi.org/10.1109/COMST.2019.2891891
2. Amado, J. R., Pereira, F. C., Signorello, S., Correia, M., Ramos, F. Poster: In-Network ML Feature Computation for Malicious Traffic Detection. Proceedings of the ACM SIGCOMM 2023 Conference, 2023, 1105-1107. https://doi.org/10.1145/3603269.3610866
3. Anderson, H. S., Roth, P. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. arXiv preprint, 2018, arXiv:1804.04637. https://arxiv.org/abs/1804.04637
4. Applebaum, A., Miller, D., Strom, B., Korban, C., Wolf, R. Intelligent, Automated Red Team Emulation. Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC), 2016, 363-373. https://doi.org/10.1145/2991079.2991111
5. Atomic Red Team. Atomic Red Team. https://www.atomicredteam.io/. Accessed on May 19, 2026.
6. Cao, X., Wang, J., Wu, P., Fang, Z. VulMPFF: A Vulnerability Detection Method for Fusing Code Features in Multiple Perspectives. IET Information Security, 2024, 2024(1), 4313185. https://doi.org/10.1049/2024/4313185
7. Chen, T., Song, Q., Zhu, T., Qiu, X., Zhu, Z., Lv, M. Kellect: A Kernel-Based Efficient and Lossless Event Log Collector for Windows Security. Computers & Security, 2025, 150, 104203. https://doi.org/10.1016/j.cose.2024.104203
8. Cheng, W., Yuan, Q., Zhu, T., Chen, T., Ying, J., Zheng, A., Ma, M., Xiong, C., Lv, M., Chen, Y. TagAPT: Towards Automatic Generation of APT Samples with Provenance-Level Granularity. IEEE Transactions on In-

formation Forensics and Security, 2025. https://doi.org/10.1109/TIFS.2025.3557742

9. Cisco. Anticipating the Unknowns. https://ebooks.cisco.com/story/anticipating-unknowns/. Accessed on May 19, 2026.
10. DARPA. Transparent Computing Engagement. https://www.darpa.mil/program/transparent-computing/. Accessed on May 19, 2026.
11. De Pasquale, G., Grishchenko, I., Iesari, R., Pizarro, G., Cavallaro, L., Kruegel, C., Vigna, G. ChainReactor: Automated Privilege Escalation Chain Discovery via AI Planning. Proceedings of the 33rd USENIX Security Symposium, 2024, 5913-5929. https://www.usenix.org/conference/usenixsecurity24/presentation/de-pasquale
12. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A. Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware. IEEE Transactions on Information Forensics and Security, 2021, 16, 3469-3478. https://doi.org/10.1109/TIFS.2021.3082330
13. Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., Rass, S. PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing. Proceedings of the 33rd USENIX Security Symposium, 2024, 847-864. https://www.usenix.org/conference/usenixsecurity24/presentation/deng
14. Du, M., Li, F., Zheng, G., Srikumar, V. DeepLog: Anomaly Detection and Diagnosis from System Logs Through Deep Learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017, 1285-1298. https://doi.org/10.1145/3133956.3134015
15. Du, M., Luu, A. T., Ji, B., Wu, X., Huang, D., Zhuo, T. Y., Liu, Q., Ng, S.-K. CodeArena: A Collective Evaluation Platform for LLM Code Generation. arXiv preprint, 2025, arXiv:2503.01295. https://arxiv.org/abs/2503.01295
16. Edge, M. E., Falcone Sampaio, P. R. A Survey of Signature Based Methods for Financial Fraud Detection. Computers & Security, 2009, 28(6), 381-394. https://doi.org/10.1016/j.cose.2009.02.001
17. Fu, Z., Liu, M., Qin, Y., Zhang, J., Zou, Y., Yin, Q., Li, Q., Duan, H. Encrypted Malware Traffic Detection via Graph-Based Network Analysis. Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses (RAID), 2022, 495-509. https://doi.org/10.1145/3545948.3545983
18. Gehani, A., Tariq, D. SPADE: Support for Provenance Auditing in Distributed Environments. Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2012, 101-120. https://doi.org/10.1007/978-3-642-35170-9_6
19. Gong, C., Li, Z., Li, X. Information Security Based on LLM Approaches: A Review. arXiv preprint, 2025, arXiv:2507.18215. https://arxiv.org/abs/2507.18215
20. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M. UNICORN: Runtime Provenance-Based Detector for Advanced Persistent Threats. arXiv preprint, 2020, arXiv:2001.01525. https://arxiv.org/abs/2001.01525 https://doi.org/10.14722/ndss.2020.24046
21. Happe, A., Cito, J. Getting Pwn'd by AI: Penetration Testing with Large Language Models. Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 2023, 2082-2086. https://doi.org/10.1145/3611643.3613083
22. Hossain, M. N., Milajerdi, S. M., Wang, J., Eshete, B., Gjomemo, R., Sekar, R., Stoller, S., Venkatakrishnan, V. N. SLEUTH: Real-Time Attack Scenario Reconstruction from COTS Audit Data. Proceedings of the 26th USENIX Security Symposium, 2017, 487-504. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hossain
23. Hu, Z., Beuran, R., Tan, Y. Automated Penetration Testing Using Deep Reinforcement Learning. Proceedings of 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2020, 2-10. https://doi.org/10.1109/EuroSPW51379.2020.00010
24. Jin, J., Zhu, T., Yuan, Q., Chen, T., Lv, M., Zheng, C., Mei, J.-P., Pan, X. PDCleaner: A Multi-View Collaborative Data Compression Method for Provenance Graph-Based APT Detection Systems. Computers & Security, 2025, 152, 104359. https://doi.org/10.1016/j.cose.2025.104359
25. Kerr, D. Red Team Automation. https://github.com/endgameinc/rta. Accessed on May 19, 2026.
26. Koziolek, H., Grüner, S., Hark, R., Ashiwal, V., Linsbauer, S., Eskandani, N. LLM-Based and Retrieval-Augmented Control Code Generation. Proceedings of the 1st International Workshop on Large Language Models for Code, 2024, 22-29. https://doi.org/10.1145/3643795.3648387
27. Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., Weber, D., Webster, S. E.,

Wyschogrod, D., Cunningham, R. K., et al. Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation. Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX 2000), 2000, 2, 12-26. https://doi.org/10.1109/DISCEX.2000.821506

28. Manzoor, E., Milajerdi, S. M., Akoglu, L. Fast Memory-Efficient Anomaly Detection in Streaming Heterogeneous Graphs. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, 1035-1044. https://doi.org/10.1145/2939672.2939783
29. Mikolov, T., Chen, K., Corrado, G., Dean, J. Efficient Estimation of Word Representations in Vector Space. International Conference on Learning Representations (ICLR), 2013. arXiv:1301.3781. https://arxiv.org/abs/1301.3781
30. Milajerdi, S. M., Eshete, B., Gjomemo, R., Venkatakrishnan, V. N. Poirot: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS), 2019, 1795-1812. https://doi.org/10.1145/3319535.3363217
31. Milajerdi, S. M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V. N. Holmes: Real-Time APT Detection Through Correlation of Suspicious Information Flows. Proceedings of 2019 IEEE Symposium on Security and Privacy (SP), 2019, 1137-1152. https://doi.org/10.1109/SP.2019.00026
32. MITRE. MITRE ATT&CK. https://attack.mitre.org/. Accessed on May 19, 2026.
33. Rapid7. Metasploit - Penetration Testing Software, Pen Testing Security. https://www.metasploit.com/. Accessed on May 19, 2026.
34. Schwartz, J., Kurniawati, H. Autonomous Penetration Testing Using Reinforcement Learning. arXiv preprint, 2019, arXiv:1905.05965. https://arxiv.org/abs/1905.05965
35. Sharma, A., Gupta, B. B., Singh, A. K., Saraswat, V. K. Orchestration of APT Malware Evasive Manoeuvers Employed for Eluding Anti-Virus and Sandbox Defense. Computers & Security, 2022, 115, 102627. https://doi.org/10.1016/j.cose.2022.102627
36. Wang, Q., Hassan, W. U., Li, D., Jee, K., Yu, X., Zou, K., Rhee, J., Chen, Z., Cheng, W., Gunter, C. A., et al. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. Proceedings of the Network and Distributed System Security Symposium (NDSS), 2020. https://www.ndss-symposium.org/ndss-paper/you-are-what-you-do-hunting-stealthy-malware-via-data-provenance-analysis/ https://doi.org/10.14722/ndss.2020.24167
37. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems (NeurIPS), 2022, 35, 24824-24837. https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html https://doi.org/10.52202/068431-1800
38. Xu, J., Stokes, J. W., McDonald, G., Bai, X., Marshall, D., Wang, S., Swaminathan, A., Li, Z. AutoAttacker: A Large Language Model Guided System to Implement Automatic Cyberattacks. arXiv preprint, 2024, arXiv:2403.01038. https://arxiv.org/abs/2403.01038
39. Yamaguchi, F., Golde, N., Arp, D., Rieck, K. Modeling and Discovering Vulnerabilities with Code Property Graphs. Proceedings of 2014 IEEE Symposium on Security and Privacy (SP), 2014, 590-604. https://doi.org/10.1109/SP.2014.44

This article is an Open Access article distributed under the terms and conditions of the Creative Commons Attribution 4.0 (CC BY 4.0) License (http://creativecommons.org/licenses/by/4.0/).