From Sands to Mansions: Actionable, Customizable and Causality-Preserving Cyberattack Emulation with LLM-powered Symbolic Planning

Lingzhi Wang[1[0009−0006−3901−2131]], Zhenyuan LI[2[0000−0002−7712−0292]], Yi Jiang[2[0009−0005−1126−6538]], Zhengkai Wang[2[0009−0006−1969−7364]], Xiangmin Shen[3[0009−0001−8301−7961]], Wei Ruan[2[0000−0002−5889−3054]], and Yan Chen[1[0000−0003−4103−1498]]

[1] Northwestern University, Evanston, IL 60208, USA
lingzhiwang2025@u.northwestern.edu
ychen@northwestern.edu
[2] Zhejiang University, Hangzhou, Zhejiang 310027, China
{lizhenyuan,22421062,22451237,ruanwei}@zju.edu.cn
[3] Hofstra University, Hempstead, NY 11549, USA
xiangmin.shen@hofstra.edu

Abstract. Evolving attacker capabilities demand realistic and continuously updated cyberattack emulation for threat-informed defense and security benchmarking. Towards automated attack emulation, this paper defines modular attack actions and a linking model to organize and chain heterogeneous attack tools into causality-preserving cyberattacks. Building on this foundation, we introduce Aurora: an automated cyberattack emulation system powered by symbolic planning and large language models (LLMs). Aurora crafts actionable, causality-preserving attack chains tailored to Cyber Threat Intelligence (CTI) reports and target environments, and automatically executes these emulations. Using Aurora, we generated an extensive cyberattack emulation dataset from 250 attack reports, 15 times larger than the leading expert-crafted dataset. Our evaluation shows that Aurora significantly outperforms existing methods in creating actionable, diverse, and realistic attack chains. We release the dataset and use it to evaluate three state-of-the-art intrusion detection systems, whose performance differed notably from results on older datasets, highlighting the need for up-to-date, automated attack emulation.

Keywords: Cyberattack Emulation · Symbolic Planning · Large Language Models.

1 Introduction

The continuous evolution of cyberattack technologies makes comprehensive and up-to-date attack emulation essential for benchmarking defense systems, identifying their weaknesses, and enhancing threat-informed defenses. Consequently,

Table 1. Comparison of existing efforts in attack emulation. The checkmarks indicate whether a system satisfies each property under the definitions used in this paper. Partial or manual support is not considered sufficient.

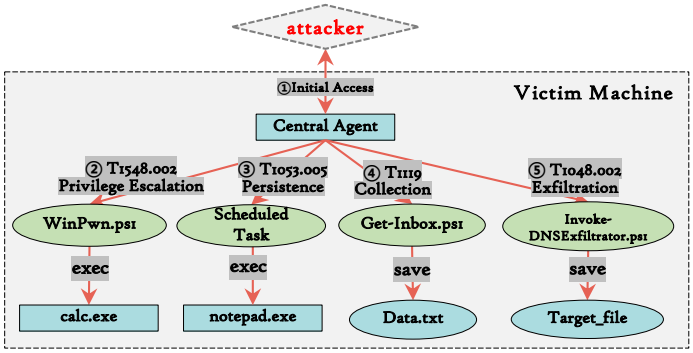
Category	Examples	Actionable	Causally Coherent	Customizable	Scalable
Human-Orchestrated Attack Campaigns	DARPA TC, OpTC	✗	✓	✗	✗
Abstract Attack Modeling	AttacKG	✗	✓	✗	✓
Single-point Testing Tools	Atomic Red Team	✓	✗	✓	✓
Procedural Attack Orchestration Tools	PurpleSharp, Caldera	✓	✗	✗	✓
Human-Crafted Attack Playbooks	MITRE Evaluation	✓	✓	✗	✗
Aurora		✓	✓	✓	✓

attack emulation has become increasingly valued by governments, organizations, and security vendors. Notable examples include the European Central Bank's threat intelligence-based ethical red teaming (TIBER-EU) framework [78], purple team exercises conducted by the U.S. Cybersecurity and Infrastructure Security Agency and the National Security Agency, and numerous commercial products [14,65,34] in this area. Since 2018, MITRE has annually organized expert emulation of advanced persistent threat (APT) attacks, attracting participation from over 50 top security vendors worldwide. The growing importance of this field is also underscored by market projections, with MarketsandMarkets [49] forecasting the automated cyberattack emulation market to expand from $729.2 million in 2024 to over $2.4 billion by 2029.

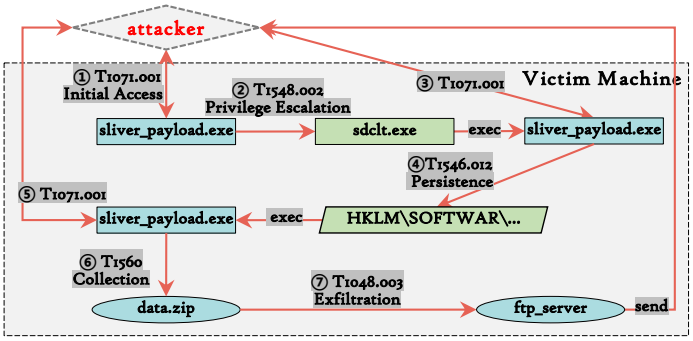
Despite growing interest, there remains a notable scarcity of *actionable*, *causally coherent*, *scalable*, and *customizable* cyberattack emulation systems, especially for multi-stage APT attacks [76,11]. An actionable attack emulation system should provide detailed commands, tools, and instructions to reproduce emulated attacks. This facilitates multi-layer data collection (e.g., network traffic, authentication events, and host-based system logs) according to specific needs. Second, effective attack emulation must preserve causal relationships between attack steps, ensuring that each step is executed based on the state and artifacts produced by previous steps, thereby accurately reflecting real-world attacker behavior. Furthermore, as new attack techniques emerge daily, a scalable emulation system must easily incorporate new techniques without architectural changes and automatically generate diverse attack chains. Finally, we consider an emulation system to be customizable if it can automatically generate attack chains that are tailored to a specific target environment or aligned with the behaviors of a particular threat actor, rather than relying on manually authored playbooks or scripts.

Table 1 highlights the limitations of existing efforts in emulating actionable, causally coherent, scalable, and customizable cyberattacks. For instance, several widely used datasets [15,74,2] are derived from human-orchestrated red

team campaigns and consist of post-incident audit logs. While these attacks are realistic and causally coherent, they are recorded only as static traces. As a result, such non-actionable datasets do not support replaying attacks for actively testing defense systems. Single-point attack emulation tools [59,32,60,23] execute isolated tests and lack mechanisms to generate multi-stage, causally coherent attack chains. While some procedural attack orchestration tools [6,67,77] attempt to assemble multi-step attacks from isolated actions, they do not explicitly model inter-step dependencies, and therefore struggle to chain attacks comparable to real-world attack campaigns (see Figure 1). Expert-crafted attack playbooks from MITRE Engenuity ATT&CK Evaluations [19] offer high-fidelity, causally coherent attack representations, but their reliance on manual expertise limits their adaptability and scalability.



(a) Each step is executed isolatedly without causality and rely on a central control agent (e.g. Caldera Agent).



(b) More realistic multi-step attacks with each step causality-connected.

Fig. 1. Causality in emulated attack chains

Three core challenges hinder the development of ideal cyberattack emulation systems. The first challenge lies in the unstructured nature of cyberattack knowledge, which requires significant human expertise to integrate diverse Tactics, Techniques, and Procedures (TTPs) from heterogeneous sources into

coherent attack chains. Consequently, many existing approaches cover only a small subset of TTPs; for instance, the state-of-the-art automated attack planner ChainReactor [64] primarily focuses on privilege escalation and considers merely about 30 attack actions. The second challenge lies in constructing multi-step and causality-preserving attack chains, which requires understanding the relationship between different attack steps. Existing studies [3,64,30] have yet to propose a formal framework for modeling these relationships. The third challenge is tailoring attacks to specific environments and threat intelligence, which ensures emulations both reflect real-world adversary behavior from CTI reports while remaining compatible with the tools and constraints of the target environment. To our knowledge, transforming CTI into attacks that can actually be executed in real environments remains an open problem.

To address these challenges, we propose AURORA, the first automated, causality-preserving, customizable cyberattack emulation system. AURORA converts third-party attack tools into standard, modular attack actions using LLM. These actions are then interconnected into causality-preserved attack chains with symbolic planning. AURORA can customize attack chains based on the emulation environment, available tools, and intelligence derived from CTI reports. It also provides actionable Python scripts to execute these attacks automatically. Experimental results demonstrate that AURORA's superior capability in building actionable, diverse, and causality-preserved attacks compared to existing approaches. AURORA has extracted over 5,500 attack actions from five popular third-party attack tools. Leveraging these actions, we constructed attacks based on 250 CTI reports. These attacks are published as a continually growing cyberattack dataset for future research.

In this paper, we make the following contributions. First, we define attack actions to modularize heterogeneous attack tools and propose a novel attack action linking model to connect them into coherent attack chains. Second, we introduce AURORA, the first automated, customizable, and causality-preserving cyberattack emulation system. Third, experiments show that AURORA generates more causality-preserved attack chains than other automated approaches and offers greater scalability and customization than expert-crafted attacks. We use the generated attacks to evaluate three state-of-the-art intrusion detection systems, whose performance differed notably from results on older datasets. To facilitate future research, we open-sourced the attack dataset generated by AURORA[4], which consists of over 250 unique attack chains, 15 times larger than the leading expert-crafted equivalent.

2 Background and Related Work

2.1 Cyberattack Emulation

The need for open, comprehensive, and continuously updated cyberattack emulation is widely recognized by both industry and the research community [11,76].

[4] `https://github.com/LexusWang/Aurora-demos`

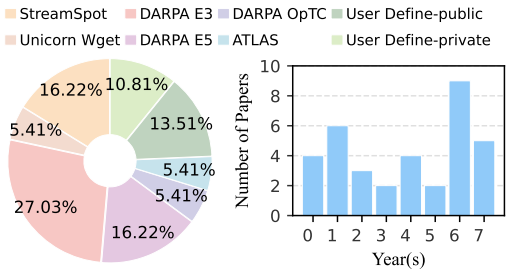


Fig. 2. Benchmark datasets in intrusion detection research (Left: Proportional usage of datasets. Right: Dataset age at the time of paper publication.)

We reviewed representative prior efforts in cyberattack emulation, including papers published in major security venues, as well as widely adopted open-source tools in the industry. Specifically, we survey (1) benchmark datasets commonly adopted in recent intrusion detection papers, (2) abstract attack modeling approaches focusing on high-level attack representation, (3) executable public cyberattack testing and emulation tools, and (4) expert-curated attack emulations used as industry benchmarks. However, no existing work satisfies our needs. This gap is reflected by the limited and outdated cyberattack benchmark datasets in recent intrusion detection papers [70,43,10,38,25,79,85,86,50,35,27,51,28,80,39]. Our analysis (shown in Figure 2) reveals two critical problems. First, the benchmarking is dominated by a few non-scalable datasets [15,74,2], indicating a narrow and repetitive evaluation scope. Second, detection systems are often evaluated against outdated threats.

Many researches [11,76,37,73,44,1,68,40] focus on abstract attack modeling, aiming to capture attacker behaviors at the semantic level using probabilistic models or CTI reports. Their primary goal is to model *what* attackers do, typically in terms of MITRE ATT&CK TTPs, to support threat understanding, detection, and attribution. However, such models remain inherently abstract. They operate at the level of TTP sequences and do not provide actionable specifications, such as concrete tools, commands, scripts, execution contexts, or inter-step state dependencies. As a result, they cannot be directly instantiated or replayed as executable attacks in real-world environments.

To address the lack of actionability in abstract attack models, many systems provide executable implementations of attack techniques. These systems can execute attack actions in controlled environments. However, most of these tools (e.g., Atomic Red Team [59], Metasploit [60], Viper [23], and LOLBAS [48]) execute individual techniques or actions in isolation rather than as a coherent attack campaign. Executing isolated steps is unable to emulate real-world multi-step attacks, and thus insufficient for evaluating defense systems that rely on cross-step correlation and contextual reasoning [84,51,9]. For instance, file compression is typically benign when performed in isolation, but becomes suspicious when preceded by data staging and followed by exfiltration.

To move beyond isolated testing, several attack emulation frameworks aim to construct multi-step attacks by orchestrating atomic actions into predefined

playbooks. Representative examples include Caldera [6], PurpleSharp [67], and Attack Range [77]. However, the resulting attack chains remain largely procedural. The actions are executed sequentially without explicitly modeling the causal relationships or shared objectives that connect them into a coherent attack campaign. As a result, the emulated attacks often deviate from real-world attacker behavior (see Figure 1). For example, escalating privileges solely to execute `calc.exe` serves no meaningful attack. We refer to such systems as *procedural attack orchestration frameworks*, as they focus on sequencing predefined actions rather than composing attacks based on causal dependencies. Moreover, the playbooks used by these frameworks are manually authored, and there is no mechanism for automatically generating or adapting playbooks based on different environments or attacker profiles. This reliance on human authoring fundamentally limits both the scalability and the customization of attack emulation.

Given those shortcomings, human-crafted attacks are still a reliable source for attack emulation [19,71]. MITRE ATT&CK Evaluations [19] provide one of the most well-recognized and public attack emulations. These attacks are fully actionable, logically coherent, and aligned with CTI reports. However, their generation requires immense manual effort and deep domain expertise. This is evident in the small number of scenarios produced (e.g., 17 attack chains from 15 adversary groups over eight years) and the substantial fees required for vendor participation. Table 1 shows the comparison of existing work.

2.2 Symbolic Planning

In general, a planning problem aims to find a sequence of actions that achieves a specific goal [7,18]. Specifically, a symbolic planning problem can be defined as a tuple $(\mathcal{P}, \mathcal{A}, I, G)$, where $\mathcal{P}$ is a finite set of predicates and $\mathcal{A}$ is a finite set of actions. A predicate ($p \in \mathcal{P}$) is a boolean proposition describing a specific condition of the target world, and a subset of $\mathcal{P}$ can represent a state s of the target world at some stage. I and G are two special states: I is the initial state, and G is the goal state. An action ($a \in \mathcal{A}$) is the basic unit that can change the states. It includes two features: the *preconditions*, which are the predicates that must be true before the action can be applied, and the *effects*, which are the predicates that will be true after the action is applied. A plan is a sequence of actions changing the target world from the initial state to the goal state. Symbolic planning typically employs *Planning Domain Definition Language* (PDDL), a declarative language, to define planning problems with symbolic notations.

Cyberattacks are often modeled as symbolic planning problems. However, existing work either lacks clear definitions of actions and states [66,36,47,46], or relies on overly abstract or limited action spaces [72,30,52,42,31,3] that fail to reflect real-world complexity [54]. More importantly, most existing approaches do not explicitly model how actions are causally linked through preconditions and effects. Some approaches [47,46] do not specify how one-shot planning can be applied to generate multi-stage APT campaigns, where each stage has distinct initial and goal states. The absence of a well-defined action space, a principled

linking model, and a multi-stage planning schema leaves a critical gap in planning actionable and causally coherent cyberattacks.

2.3 Assumptions and Scope

Unlike penetration testing and red teaming, which aim to discover vulnerabilities or attack paths in unknown or partially known systems, the goal of attack emulation is to construct and execute emulated attack scenarios within a known environment. Accordingly, AURORA assumes that basic information about the target testbed, including operating system versions, installed software, known vulnerabilities, and network topology, is available for emulation. Such information can be automatically collected using existing asset inventory and security assessment tools (e.g., OpenSCAP, OpenVAS, and osquery [61,62,63]). In this paper, we do not consider planning under incomplete information. Importantly, this assumption does not restrict AURORA to a single predefined testbed or static environment, as in some prior work [19,77]. Instead, AURORA supports customized attack emulation across different user environments, as long as basic system information is available (see §4.2 for details).

3 Cyberattack Planning

This section introduces two core concepts, *Attack Actions* and *Attack Action Linking Model (AALM)*. Attack actions provide modular and structured representations of actionable steps in cyberattacks. The attack action linking model defines the relationships between these actions and connects them into causality-preserved attack chains.

3.1 Attack Actions

In symbolic planning, an action is the atomic unit in a plan. Similarly, we define an attack action as an atomic operation in cyberattack emulations.

Definition 1 (Attack Action). *An attack action is the smallest unit that is considered to form an attack plan and an atomic operation to execute in a multi-step attack.*

The granularity of an attack action depends on how attack tools are invoked in practice. For example, an individual test in Atomic Red Team [59], a module in Metasploit [60], or a single command in post-exploitation frameworks [5,21] can each be treated as an attack action. As summarized in Table 2, each attack action has several associated attributes. *Name*, *source*, and *description* define the basic information of the action. *Tactics and techniques* map the action to the entries in the MITRE ATT&CK matrix. The *execution* field specifies both concrete execution instructions (e.g., commands or scripts) and the required executor, which denotes the execution context or agent. For example, a malicious PowerShell script uses `PowerShell` as its executor, whereas commands provided

by Meterpreter [69] require an active Meterpreter session on the target system and therefore designate `Meterpreter` as their executor. The *precondition* and *effect* attributes are essential for linking actions together. Therefore, they are discussed in detail in the next section on the attack action linking model.

Table 2. Attributes of attack actions

Field	Description
UUID	A universally unique identifier.
Name	A short name of the attack action.
Description	A brief description of the attack action.
Source	The source tool of the attack action.
Tactics	MITRE ATT&CK tactics.
Technique	MITRE ATT&CK technique.
Execution	The concrete executor and instructions for execution.
Preconditions	Conditions that must be satisfied for action execution.
Effects	Conditions that will be satisfied after action execution.

The concept of attack actions is related to, but distinct from, MITRE ATT&CK procedures [55]. While procedures provide descriptions of how a technique may be implemented in practice, they are primarily intended for knowledge sharing and threat intelligence documentation. As a result, procedures are expressed in unstructured natural language and do not explicitly define execution context, preconditions, or effects. In contrast, attack actions are designed as executable and plannable primitives. Each attack action explicitly specifies its executor, concrete instructions, preconditions, and effects, which enables attack actions to be automatically composed into causally coherent, actionable attack chains. For example, a MITRE procedure may state that attackers dump credentials from LSASS using tools such as Mimikatz. However, this description does not explicitly encode the prerequisite of administrative privileges or the resulting availability of credential artifacts. In our model, credential dumping is represented as an attack action whose execution requires administrator access and whose effect is the acquisition of credentials.

3.2 Attack Action Linking Model

As introduced in §2.2, preconditions and effects are used to connect actions in symbolic planning. Preconditions represent the requirements that must be met before executing an attack action. For example, executing Sliver commands requires an active session on the victim machine, and data exfiltration requires the data to be collected. Effects are the resulting conditions after executing an attack action. An action is included in an attack chain only if 1) all its preconditions are fulfilled, and 2) its effects help satisfy the preconditions of the following actions or achieve specific attack goals. However, the term *condition* is broad, subjective, and ambiguous. In this paper, we propose the Attack Action Linking Model (AALM), a standard, universal, and succinct framework, to describe the preconditions and effects from the perspective of cyberattack emulation.

Definition 2 (Attack Action Linking Model). *The attack action linking model is an extensible set of PDDL predicates used to describe the preconditions and effects of an attack action.*

We design the predicate set of AALM from nine dimensions: environments, executors, payloads, files, processes, users, information, data, and techniques. Due to space limits, the full list is available in our open-source repository.

- *Environment* includes the predicates related to the operating systems, vulnerabilities, software, and network topology of the attack target. To standardize the predicate format across numerous vulnerabilities and software types, we use Common Vulnerabilities and Exposures (CVE) [13] to identify vulnerabilities and Common Platform Enumeration (CPE) [16] to represent software.
- *Executor* includes predicates about the executor type and states. Executor type encompasses common executors derived from popular attack frameworks and C2 tools [60,59,48]. Executor states contains predicates about privilege levels, persistence status, and so on.
- *Payload* includes predicates describing payload types, formats, and operations. The payload types and formats are derived from commonly used payload generation tools [56]. Payload operation predicates cover the conditions about loading, executing, and setting handlers.
- *File* includes predicates related to file types, states, and operations. To ensure AALM covers the file types commonly involved in cyberattacks, we crawled documentation of popular open-source attack tools [59,60,48] and cybersecurity websites [23,53] to extract a representative set of file types. It also covers similar predicates for directories and Windows registry entries.
- *Information* includes predicates representing information involved in cyberattacks. Given the diversity of information types in cyberattacks, we organize them into 39 categories based on the technique types defined under the MITRE discovery tactic.
- *Data* includes the predicates representing data types and data states. We categorize ten data types according to the technique taxonomy of the MITRE collection tactics. Data state predicates include the storage, transmission, and relay of data during an attack.
- *User* includes the predicates representing user states, types, username, credentials, and privilege levels of users.
- *Process* includes the predicates representing process states and process information such as process ID, name, and privilege level.
- *Technique* includes the predicates that reflect the intent or consequence of an action. For example, although deleting backup files and deleting log files both result in (file-deleted), their purposes are different. The former aims to inhibit system recovery, while the latter serves to remove indicators of compromise. Therefore, we design technique-level predicates such as (inhibit-system-recovery) and (indicator-removal) to show the difference of the effects.

Figure 3 shows an attack chain connected by predicates in AALM. Given the complexity and evolving nature of cyberattacks, AALM is built with extensibility. New predicates and actions can be incrementally added, similar to the

ongoing expansion of the MITRE ATT&CK framework. AALM allows users to define custom actions along with their preconditions and effects using PDDL syntax. AALM follows a monotonic extension principle: newly introduced predicates are incorporated through additional action preconditions and effects. Existing attack chains remain unchanged and valid under the extended model. For example, a user can introduce a predicate `(edr-service-disabled ?h)` as the effect of an evasion action and as the precondition for high-risk payload execution, explicitly modeling the causal relationship between disabling a specific endpoint protection and enabling stealthy execution. As another example, when certain attack actions are only feasible with specific tools, users can introduce tool-specific predicates (e.g., a dedicated executor-type predicate) as the preconditions of those specialized actions. As the first work to propose a standardized, symbolic linking model for automated attack emulation, we envision its coverage and expressiveness to grow through collaboration with the broader security community.

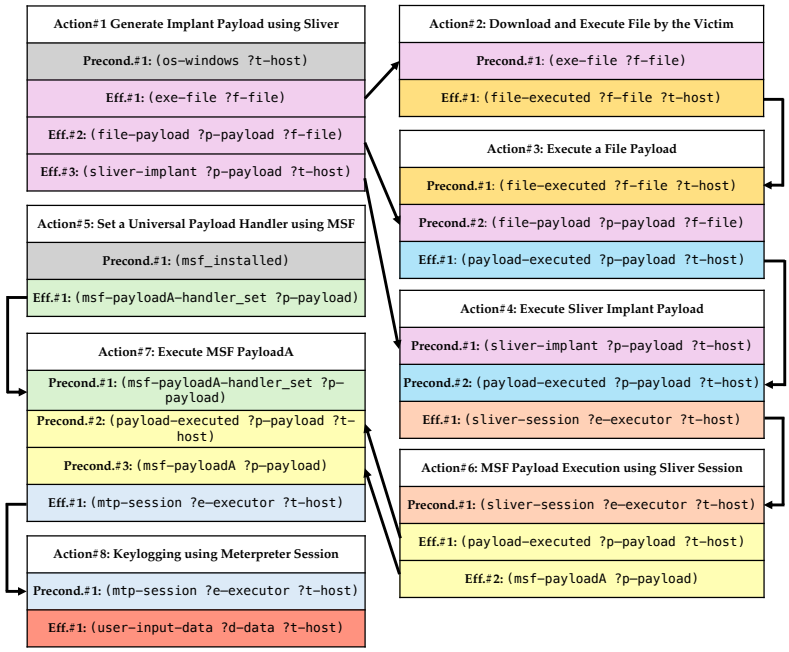


Fig. 3. An attack chain example connected by AALM.

4 System Design

Based on the attack actions and AALM, we design AURORA, an automated, customizable, causality-preserving cyberattack emulation system. As shown in

Figure 4, AURORA encompasses three stages: ①*Attack Action Formalization*, ②*Attack Planning*, and ③*Attack Execution*. AURORA first builds attack actions from existing tools. During attack planning, AURORA considers available attack actions, emulation environments, and CTI reports to formalize the planning problem in PDDL. The symbolic planner then chains the actions to a plan. Finally, a Python script is generated to execute the attack plan in the emulation environment.

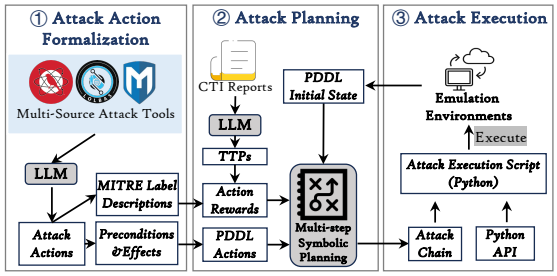


Fig. 4. System overview.

4.1 Attack Action Formalization

The emulated attacks leverage existing tools rather than zero-day exploits, ensuring that AURORA evaluates defense systems only against known attack techniques (we discussed ethical considerations in §8). AURORA reviews documentation of attack tools and converts them into structured attack actions defined in §3. However, analyzing heterogeneous documentation is challenging. Although some simple features, such as name and description, are provided in the documentation and can be extracted with regular expressions, identifying preconditions and effects requires a deeper semantic understanding and substantial domain expertise in cybersecurity.

Recent studies show that LLMs demonstrate strong capabilities in both text comprehension and domain-specific reasoning [82,29]. AURORA leverages LLMs to formalize third-party tools into attack actions. As shown in Figure 5, we begin by categorizing all tools into 14 groups based on their corresponding MITRE tactics. For each tactic, we design a prompt for LLM to analyze the preconditions and effects. The prompt consists of five components: a task description, a step-by-step thinking process (a series of questions), a set of candidate predicates from the AALM, illustrative examples of correct outputs and common mistakes, and formatting requirements for the output. For each question in the thinking process, the LLM is restricted to selecting only from the given predicate set. If no suitable predicate is available in AALM, the LLM is instructed to output *N/A*, and the action is excluded from the attack emulation. This approach minimizes hallucination and ensures that only actions compliant with AALM specifications are retained for subsequent symbolic planning.

As an example, consider tools used for payload generation, which fall under the *Resource Development* tactic, including widely used tools such as Msfvenom,

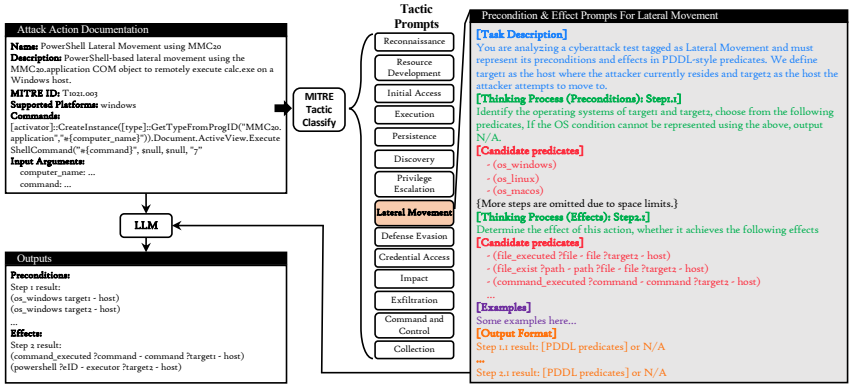


Fig. 5. An example showing how AURORA analyzes preconditions and effects of an attack action using AALM.

Sliver, and Empire. The corresponding prompt first outlines the function of these tools and the analysis task. It then details the analysis process. For preconditions, the LLM is asked to identify the target OS and architecture of the payload and selects matching predicates from AALM; if none apply, it returns N/A. For effects, the thinking process begins by checking whether the payload type matches any predefined payload types in AALM; if not, N/A is returned. Next, it determines the payload format (file, shellcode, one-liner command, raw packet). If the payload is a file, its file type is also identified. If any of these steps yield N/A, AURORA excludes the action from the action space. In our experiments, AURORA uses GPT-o3 (o3-2025-04-16) as the underlying LLM for predicate extraction. Prompts were iteratively refined during development using a small validation set for each tool category to ensure syntactic correctness. Once finalized, the prompts were fixed and not modified during evaluation.

Experimental results show that AALM and our prompt engineering can cover the majority of attack actions and formalize more than 5000 actions. AALM also addresses the problem of relying on MITRE tactic labels in attack emulation. For instance, certain tests in Atomic Red Team are labeled as *Lateral Movement*, yet their actual effect does not achieve remote file transfer or code execution. Therefore, AURORA ensures these actions are not included in the emulation solely based on their MITRE labels. In §5.2, we show how this helps avoid invalid attack emulation compared to existing work.

4.2 Attack Planning

AURORA uses symbolic planning to chain the attack actions. It represents attack actions and emulation environments in PDDL, which is then consumed by an off-the-shelf symbolic planner to generate execution plans. Moreover, we designed a reward function mechanism to customize attack emulation based on CTI reports and a multi-step planning process to generate multi-stage attack emulation plans.

Representing Emulation Environments in PDDL The emulation environment, i.e., the target machine(s) of the attack, determines which actions are feasible. Therefore, the starting state I in attack planning should reflect the initial state of the emulation environment. Prior to emulation, basic information about the target machine(s) is required, including operating systems, known vulnerabilities, installed software, and domain or network configurations. This information can be easily obtained by running automated asset-inventory tools such as OpenSCAP, OpenVAS, and osquery [61,62,63]. AURORA supports parsing the outputs of these tools and automatically map them to the environment predicates in AALM. In §5.4, we will show how AURORA generates customized attacks based on different emulation environments.

Reward Function for CTI-based Customization AURORA aims to emulate customized cyberattacks tailored to the characteristics of specific threat groups. To achieve this, AURORA leverages CTI reports, which document the behaviors of real attackers. We design the reward function to generate attack chains that closely resemble CTI reports. We first extract TTPs from the reports. Despite various approaches in analyzing CTI reports [37,73,44,1,68,40], our experiment in A.2 shows that LLMs surpass these methods in terms of the accuracy. Therefore, we employ off-the-shelf LLMs to extract TTPs from the reports. To mitigate the hallucination issue and standardize the output, we include MITRE ATT&CK TTP definitions and an output template in the prompt. The improvement of CTI report analysis is not the focus of this paper.

AURORA customizes attack emulation based on CTI reports by generating attack chains that best match the TTPs extracted from the report. To accomplish this, we assign a reward to each attack action. Attack actions whose TTPs appear in the report are assigned higher rewards, while others receive lower rewards. By doing so, maximizing the total reward of an attack chain directly corresponds to maximizing its similarity to the CTI report. To compute the reward of an action, we first check whether its MITRE label appears in the TTP set extracted from the report. If not, the reward is set to zero. Otherwise, the reward is computed as the cosine similarity between the embeddings of the action description and that of the corresponding TTP description. Therefore, actions with more similar semantics will receive higher rewards. For example, consider two privilege escalation actions: bypassing User Account Control (UAC) and process injection. While both actions can elevate the attacker's privileges, if the report contains some phrases such as *abusing the UAC*, the former action will have a higher reward. If two attack chains are generated in the same emulation environment, the one with the higher reward is selected. Please note that we deliberately chose cosine similarity for its simplicity, efficiency, and ease of integration into planning. More sophisticated semantic alignment functions (e.g., cross-encoders or entailment-based models) can be easily plugged in. The evaluation in §5.4 shows that the cosine-based rewards yield over 30% improvement in CTI-level customization.

Multi-step Symbolic Planning Setting the goal states of the planning problem remains challenging for the multi-step attacks. On one hand, setting a single predicate as the goal state results in short chains that achieve only one attack objective. However, if we combine all desired predicates into a single goal state, any inaccessible predicate causes the entire planning process to fail. To address this, we designed a multi-step planning procedure. Specifically, we created 12 pseudo actions, which correspond to 12 high-level attack stages defined by MITRE tactics. The preconditions of a pseudo action are predicates that indicate the success of a given stage, connected by a logical *or*. The effect of each pseudo action is a dummy predicate that serves as the planning goal for that stage. The symbolic planner then searches for a feasible attack chain that can achieve this goal. When multiple chains exist, AURORA selects the one with the highest reward. Pseudo action provides a concise and clear way to organize attack stages in symbolic planning. The multi-step planning is then conducted according to the order of the APT lifecycle: initial access, execution, discovery, persistence, privilege escalation, defense evasion, credential access, lateral movement, collection, command and control, exfiltration, and impact. The goal state of each step is the dummy effect predicate of the pseudo action. The initial state before the first attack stage consists of the emulation environment predicates. After completing a stage, AURORA collects the effect predicates of the newly selected actions and adds them to the initial state. With the updated initial state, AURORA proceeds to plan the attack path for the next stage. For instance, after completing the execution stage, we obtain (sliver_session ?e-executor ?t-host) as one of the effect predicates. This predicate is then added to the initial state for planning the next stage. Note that the attack stage order is customizable. For instance, sabotage-focused attacks may skip data collection and exfiltration stages. Users can also reorder certain stages, as the predefined planning sequence does not constrain the causal dependencies between actions. For example, if persistence requires privilege escalation as a precondition, AURORA will automatically place privilege escalation actions earlier, even if their stage comes later in the original planning sequence. Multi-step planning of AURORA maximizes attack tactic coverage and stays robust when some stages are infeasible.

Handling Execution Failures: AURORA explicitly handles execution-time failures during attack emulation. When an action fails to execute due to unexpected runtime errors, AURORA removes the failed action from the action space and triggers re-planning for the current attack stage. Since attack campaigns are decomposed into 12 stages, execution failures are isolated within the current stage and do not affect previously completed stages. In other words, re-planning is restricted to the stage at which the failure occurs. For example, if a PowerShell-based data collection action fails during execution, AURORA excludes this action from the action space and re-plans the data collection stage using the remaining candidate actions. If no alternative action sequence can achieve the objective of the current stage, AURORA skips the stage and proceeds to plan subsequent stages. This design ensures that execution failures do not halt the entire attack

emulation process and allows AURORA to adaptively recover from runtime errors without requiring global rollback.

4.3 Attack Execution

The final step is to execute the generated attacks in the emulation environment. Different from existing work [64,3,52,31], AURORA can generate actionable attack emulation plans since each attack action corresponds to a concrete instruction from a tool. To further enhance automation, we developed a Python function API covering all tools integrated in AURORA, including running Metasploit modules, executing commands in the Sliver console, executing commands on the attacker machine, and simulating simple user actions. Based on the action sequence produced by the planner, AURORA automatically generates a Python script using this API. This allows an entire attack chain to be executed by simply running a script. In §5.6, we collected system logs when executing emulated attacks and used them to test three advanced intrusion detection systems.

5 Evaluation

We evaluate the performance of AURORA along five key dimensions. Specifically, we assess whether AURORA can (1) successfully emulate attacks, (2) generate causally coherent attack chains, (3) customize attack emulation based on target environments and CTI reports, (4) improve the scale and diversity of attack emulation, and (5) support the evaluation and benchmarking of security defenses. We present a case study to further illustrate how AURORA helps generate a complete, causality-preserved attack chain. More evaluations, including the costs of AURORA and the accuracy of TTP extraction, are presented in Appendix A.2.

5.1 Evaluation Setup

Attack Tools. AURORA converts existing attack tools to attack actions. In this paper, we utilize Atomic Red Team [58], Metasploit [60], Meterpreter [69], Sliver [5], and MSFvenom [56]. They represent typical cyberattack tooling: Metasploit and MSFvenom cover common pre-exploitation tasks, Atomic Red Team supplies the post-exploitation library, and Meterpreter and Sliver are mature real-world tools. The above attack tools provide AURORA with 5,555 attack actions, more than many other automated cyberattack systems [17,64].
CTI Reports. We collected 250 CTI reports from multiple sources [12,75,44]. The CTI report dataset used in our evaluation will be open-sourced.
Implementations. We implemented AURORA with 4K LoC in Python. We used the GPT models from OpenAI and leveraged Fast Downward [4] to solve symbolic planning problems. We set up the attack emulation environment infrastructure using Oracle VirtualBox, creating a small-scale LAN with 15 different hosts. We release the code, datasets, and results to the public to facilitate further research.

Baselines. As shown in Table 1, the baselines can be divided into four categories. The first category is human-crafted cyberattack emulation datasets, including MITRE Evaluation [19], and APT Attack Simulation [71], a popular open-source attack sets. The second category consists of procedural attack orchestration frameworks, including Caldera [6], Attack Range [77], and PurpleSharp [67]. The third category is the single-point testing tools, including Atomic Red Team [59], Metasploit [60], and Viper [23]. The last category is the abstract attack modeling studies [31,3,52,64]. Specifically, when evaluating the success rate and causality preservation of attacks, we compared AURORA against PurpleSharp, Caldera, and the MITRE Evaluation because they provide modularized attacks, which are well-structured and suitable for quantitative analysis.

5.2 Success Rate of Emulated Attacks

We first evaluated whether the attacks could successfully achieve malicious objectives. We examined the main objectives of cyberattacks [45,57], including espionage, data theft, sabotage, and gaining a long-term strategic advantage, and mapped them to six MITRE tactics: persistence, privilege escalation, collection, exfiltration, impact, and lateral movement. Note that the remaining tactics, such as initial access and execution, primarily correspond to intermediate stages of an attack. We then evaluated how many emulated attacks successfully accomplished these tactics. Specifically, for privilege escalation, we assessed whether the emulated attack gained elevated privileges (e.g., root or sudo on Unix systems, or Administrator and System on Windows). For persistence, we evaluated whether the attack established persistent access by maintaining a C2 agent or obtaining valid credentials. For collection, we verified whether the attack acquired user data. For exfiltration, we checked whether the attack transmitted collected data back to the attacker. For lateral movement, we measured whether the attack successfully moved laterally to another machine by executing a file, command, or script on a second host. For impact, we assessed whether the attack caused any destructive effects, including deleting user files, removing access, stopping services, or shutting down the system. The results in Table 3 show that AURORA generates almost 10 times more attacks than the baseline and achieves the highest success rate across all tactics. We analyzed the attack chains and identified two main reasons for the superiority of AURORA. First, AURORA uses AALM to link attack actions, rather than relying on MITRE labels to arrange actions. This avoids the invalid step due to the mismatch between MITRE labels and the actual action effects. For example, in MITRE's Fin7 emulation [20], *Dump SAM via Mimikatz(T1003.002)* was used for privilege escalation. However, executing this step does not directly provide the attacker with administrative privileges. It merely dumps SAM credential data; attackers must take additional steps to extract plaintext credentials or hashes from it and then leverage those to gain high-privilege access. On the contrary, AURORA ensures the high-privilege access after the privilege escalation step because AALM provides more concrete and accurate definitions of effects, closely aligned with actual attack emulation, which enhances the success rate and realism of emulations. Second, multi-step

planning allows AURORA to integrate multiple malicious behaviors into a single attack chain. In contrast, most attack chains from tools like PurpleSharp and Caldera focus on just one tactic. With multi-stage planning, users can freely determine which tactics appear in the chain, enabling broader coverage of malicious behaviors within a single emulation scenario.

Table 3. Comparison of success rates(with 95% confidence intervals).

	PurpleSharp	Caldera	MITRE Eval.	AURORA
# Total Chains	23	28	8	**250**
Persistence	8.7% (1.5,27.0)	0 (0.0,12.3)	87.5% (47.4,99.7)	**95.6%** (92.3,97.8)
Privilege Esc.	4.4% (0.1,22.0)	3.6% (0.1,18.4)	50.0% (15.7,84.3)	**100.0%** (98.5,100.0)
Collection	0 (0.0,14.8)	39.3% (21.5,59.4)	62.5% (24.5,91.5)	**98.0%** (95.4,99.4)
Exfiltration	0 (0.0,14.8)	28.6% (13.2,48.7)	37.5% (8.5,75.5)	**100.0%** (98.5,100.0)
Impact	0 (0.0,14.8)	7.1% (0.9,23.5)	37.5% (8.5,75.5)	**59.2%** (52.8,65.4)
Lateral Move.	30.4% (13.2,52.9)	28.6% (13.2,48.7)	87.5% (47.4,99.7)	**100.0%** (98.5,100.0)

5.3 Causality in Emulated Attacks

AURORA aims to emulate high-fidelity attacks that are not only successful, but also preserve causal relationships between actions. For example, it is not sufficient to merely emulate successful data collection and exfiltration in isolation. Additionally, what is exfiltrated should be exactly what was collected. Similarly, the process that gains elevated privileges should be the one used in subsequent actions. As illustrated in Figure1(a), although all four steps are completed successfully, they are not causally linked. To quantify the causality in an attack chain, we assessed logical connections between steps from three dimensions: 1) *Executor Establishment*: Whether preceding steps establish the executor to run the current step. 2) *Argument Sharing*: Whether the step shares arguments (e.g., hostnames, file paths) with other steps in the chain. 3) *Artifact Interaction*: Whether the step utilizes or affects system artifacts (e.g., files, processes) that are involved in other steps. Correspondingly, we define three metrics: *Executor Traceability Rate*, *Shared Arguments Rate*, and *Shared Artifacts Rate*. Each metric measures the proportion of steps in a chain satisfying the corresponding criterion. In addition, we evaluated the average length of the attack chains.

We verified attack chains from the baselines and AURORA, with the results summarized in Table 4. Our analysis highlights three key findings. First, AURORA and the MITRE Evaluations provide significantly longer attack chains than PurpleSharp and Caldera. Longer chains can better reflect the nature of multi-step APT attacks and offer more opportunities for detection. Please note that AURORA can generate even longer chains if increasing the number of actions per stage during the multi-step planning. Second, AURORA and MITRE Evaluations can construct full-lifecycle attack chains. We define a full-lifecycle attack as one commencing with initial access or execution and including at least one remaining downstream tactic. In contrast, the chains from PurpleSharp and Caldera typically focus on a single tactic. Finally, in terms of causality preserving, AURORA significantly outperforms PurpleSharp and Caldera, achieving comparable

performance to the expert-crafted MITRE chains. The executor traceability for both PurpleSharp and Caldera is zero. This is an expected outcome of their common architecture: a sequence of isolated scripts orchestrated by a central control agent, without modeling how that agent is initially established on the victim machine. In addition, the high rates of shared arguments and artifacts in AURORA's chains indicate strong internal connectivity.

Table 4. Comparison of causality preservation.

	Avg. Steps	Full Lifecycle	Causality-preservation Executor Trac.	Shared Args	Shared Artifact
PurpleSharp	3.2	✗	0	0	20.3%
Caldera	5.4	✗	0	15.1%	54.6%
MITRE Eval.	53.6	✓	94.4%	16.7%	93.4%
AURORA	**55.2**	✓	**100%**	**40.0%**	**99.4%**

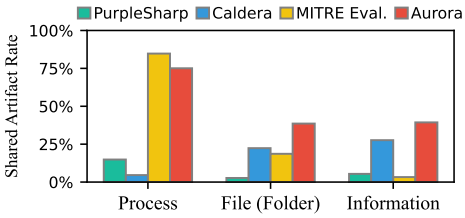


Fig. 6. Comparison of shared artifact rates.

To further analyze how AALM helps preserve causality, we examined the types of shared artifacts that connect steps in the emulated attack chains. As shown in Figure 6, chains generated by AURORA exhibit a higher proportion of steps linked via shared files and information, while the MITRE chains show more connections through shared processes. This is because AURORA models pre-foothold stages such as payload generation and initial access, where causal links are typically established via artifacts like files and target information. In contrast, MITRE chains focus more on the stages after a foothold has already been established, where causality is more likely to manifest through process-level interactions. Next, we investigated the contribution of different AALM predicate categories in preserving causality, since the linking relationship expressed by some categories in AALM (e.g., those for operating systems, CVEs, or exploit-payload mappings) has been considered in prior attack planning work [66,36,8,24]. Our findings show that the novel predicate categories in AALM, such as those related to executors, payloads, files, information, users, and data, are crucial: they constitute 70.1% of all predicate connections in the generated attack chains.

5.4 Customization of Attack Emulation

Customization based on CTI Reports We designed the reward function to prioritize the attack actions mentioned in the CTI reports during planning

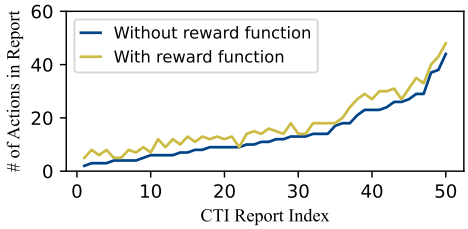


Fig. 7. Fidelity of emulated attack chains to source CTI reports.

to achieve CTI-level customization. For this evaluation, we selected 50 CTI reports and generated two attack chains for each: a report-guided chain, using the reward functions during planning, and a report-agnostic chain created from the same action space without CTI guidance. We then compare the number of CTI-mentioned techniques in the generated attacks. Across 50 CTI reports, incorporating the reward function during planning increases the number of CTI-mentioned techniques from 13.02 to 16.98 on average (+30.5%). A paired t-test shows the improvement is statistically significant.

We also present a case study, a CISA report on Phobos Ransomware, to show how the reward function mechanism guides the planner toward report-aligned attacks to achieve report-level customization. In the initial access stage, the report notes that *"...threat actors send spoofed email attachments that are embedded with hidden payloads such as SmokeLoader..."*. Therefore, AURORA selects the action of delivering payloads via email attachments (T1566.001) as malicious files (T1204.002) over other alternatives like using Metasploit exploit modules or supply chain attacks. For the persistence, the report states that *"...Phobos has also been observed using Windows Startup folders and Run Registry Keys to maintain persistence..."*. Accordingly, AURORA favors the action that modifies the user shell folder startup value (T1547.001) over other options, such as creating a new account (T1136) or abusing boot or login initialization scripts (T1037). Please note that AURORA balances CTI fidelity with actionability and causality. When a tool mentioned in the report, like SmokeLoader, is unavailable, it substitutes a close alternative (e.g., Sliver) and arranges the subsequent actions automatically to preserve the attack chain's logic.

Customization based on Emulation Environments We demonstrate AURORA's capability of tailoring attacks to specific user environments through a case study where AURORA generates attack plans for two distinct environments: a modern Linux system (CVE-2021-3493) and an older Windows system (CVE-2015-2342). For both environments, the objective was to create a five-stage attack plan covering initial access, execution, discovery, privilege escalation, and persistence. By simply modifying the initial state predicates in the respective PDDL problem files, AURORA's symbolic planner automatically selected compatible attack actions from our linking model to generate a unique chain for each environment. Figure 8 illustrates these results, showing two attack chains that achieve the same high-level goals using entirely different, environment-specific implementations. Importantly, AURORA does not fail to generate attack chains

Table 5. Comparison of cyberattack dataset scale and TTP diversity.

	Attack Chains	Attack Actions	MITRE Techniques	MITRE Tactics
APT Simulation	13	N/A[1]	N/A[1]	N/A[1]
MITRE Eval.	16	619	152	12
PurpleSharp	25	181	30	8
Caldera	28	1530	309	13
Atomic Red Team	N/A[2]	1568	309	13
Viper	N/A[2]	97	N/A[3]	10
SVED	N/A[2]	4000+	262	12
Lore	N/A[3]	N/A[3]	68	10
ChainReactor	N/A[3]	31	N/A[3]	N/A[3]
AURORA	250	5555	327	14

[1] No modularized steps that can be mapped to MITRE ATT&CK TTPs.
[2] Cannot construct multi-step cyberattacks.
[3] Not clearly reported.

due to the absence of environment-specific actions. When certain specialized actions are unavailable, AURORA can always fall back to composing attack chains using more generic and widely applicable actions, ensuring that valid multi-stage attack plans can still be generated for a given environment.

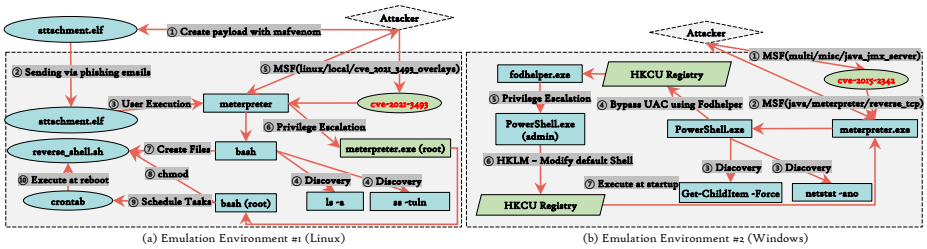


Fig. 8. Customized attacks for different environments.

5.5 Diversity of Emulated Cyberattacks

This section evaluates the scale and diversity of the attack chains generated by AURORA. We assess scale by the total number of generated attack chains and diversity by the number of unique attack actions, MITRE ATT&CK tactics, and techniques covered. Our comparison includes a wide range of baselines. As detailed in Table 5, AURORA surpasses all baselines in both scale and diversity. In terms of scale, AURORA generates a significantly higher number of unique attack chains. This is a direct result of our approach, which defines and links modular attack actions, allowing for the creation of numerous valid chains through meaningful permutations. In terms of diversity, AURORA covers the largest number of unique actions, techniques, and tactics. This breadth is achieved through its ability to integrate and leverage a wide array of third-party attack tools from different sources.

Table 6. Comparison of detection performance against attacks generated by AURORA (with 95% bootstrap confidence intervals).

	NODOZE	PROVDETECTOR	FLASH
True Positive Rate	18.92% (10.04,31.71)	48.16% (29.61,60.14)	9.67% (2.60,17.99)
False Positive Rate	2.35% (1.19,4.18)	1.72% (0.92,2.58)	0.06% (0.01,0.12)

5.6 Evaluating Existing Detection Systems

In this section, we used 250 attacks generated by AURORA to evaluate three advanced, open-source attack detectors, namely, NODOZE [28], PROVDETECTOR [81], and FLASH [70]. During these emulations, system traces were collected via ETW and converted into provenance graphs. We established ground truth by labeling system entities involved in the attacks. For training data, we simulated five hours of benign user activity. All three detectors perform anomaly detection on provenance graphs, but use different techniques. We evaluated the true positive rates (TPR) and false positive rates (FPR). For each metric, we first apply the Friedman test [22] to assess overall differences among detectors. When significant, we conduct post-hoc pairwise comparisons using the Wilcoxon signed-rank test with Holm correction ($\alpha = 0.05$) [83,33].

Table 6 shows the detection results. The Friedman test indicates significant overall differences among detectors in both TPR ($p = 0.0046$) and FPR ($p = 0.0011$). For TPR, our testing shows that PROVDETECTOR and NODOZE significantly outperform FLASH. PROVDETECTOR also achieves higher TPR than NODOZE, although the difference is not statistically significant. For FPR, post-hoc analysis shows that FLASH achieves a significantly lower false positive rate than both PROVDETECTOR and NODOZE, while the latter two are statistically indistinguishable. In general, we observed a significant performance drop on our dataset compared to those used in prior work, likely due to the greater sophistication of attacks generated by AURORA. We identified two primary reasons for this decline. First, FLASH's GNN has only two hidden layers, limiting its receptive field to two-hop neighbors. This may be sufficient for short attack paths, but inadequate for longer, causality-preserved attack chains from AURORA. Capturing such paths would require a deeper GNN architecture. Similarly, PROVDETECTOR selects the K rarest paths for anomaly detection. However, since AURORA generates longer attack chains with more steps, the original K setting is insufficient to cover all attack-related paths, leading to missed detections. Second, AURORA heavily utilizes living-off-the-land (LotL) techniques, which blend in with benign behavior and may further reduce the effectiveness of anomaly-based detectors, particularly NODOZE, which uses event frequency for detection. Although a comprehensive benchmarking study lies outside the scope of this paper, this experiment demonstrates the necessity of building more frequently updated attack datasets.

5.7 Case Study

We present a case study on a recent real-world APT group active since 2024, Silver Fox [26,41], to further illustrate how AURORA and AALM help generate a complete, causality-preserved attack chain. At present, our understanding of the group remains limited. Only a few isolated attack techniques are identified, which are insufficient for constructing attack emulations to support testing or data collection. In collaboration with a red team, we obtained three steps used by the Silver Fox attack, and manually assigned preconditions and effects according to AALM. These three steps focus solely on the tactics of execution, persistence, and impact. Without AALM, it would be hard and time-consuming to integrate them with existing actions to build a complete APT attack emulation.

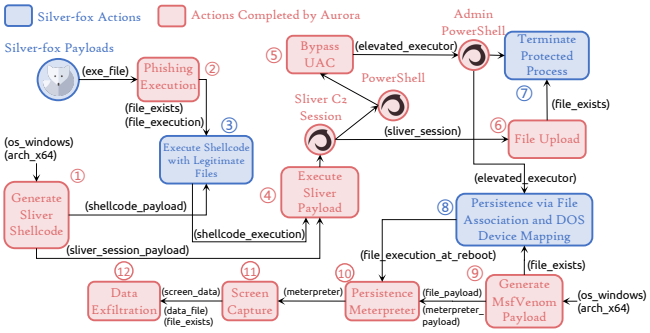


Fig. 9. The case study showing how AURORA constructs a complete attack chain for three "Silver Fox" attack actions.

The attack chain shown in Figure 9 contains the three actions from Silver Fox and nine actions added by AURORA to form a complete, causality-preserved attack. The first action shows Silver Fox is abusing legitimate, digitally signed programs to execute malicious code via techniques like DLL hijacking. With (file_executed) and (shellcode_payload) as preconditions and (shellcode_executed) as the effect, AURORA automatically built this action into the attack chain by linking it with phishing attachment execution and Sliver shellcode generation. The second action exploits a known vulnerability in a legitimate, signed driver (wsftprm.sys) to gain kernel-level privileges. This action requires the vulnerable driver file and the attacker executable to be placed (file_exists), and a PowerShell with administrative privileges (powershell) and (elevated_executor). AURORA integrates it into a complete attack chain that uses Sliver to upload malicious files and perform a UAC bypass to obtain administrative privileges. The third action of Silver Fox establishes stealthy persistence by chaining several evasive maneuvers. The precondition requires an administrative PowerShell environment to execute the script, represented by (powershell) and (elevated_executor). The effect of the action is (file_execution_at_reboot). The administrative PowerShell obtained for the previous action is reused. This action is then used to establish a persistent Meterpreter session for subsequent data exfiltration.

5.8 Ablation Study

We evaluate the effectiveness of our prompt design for predicate extraction by comparing LLM outputs against manually curated ground truth. We perform stratified random sampling of 150 actions across 14 tactic groups. Two authors independently annotate each action's preconditions and effects based on AALM, resolving disagreements through discussion. On this set, AURORA achieves high accuracy in predicate analysis. Specifically, for 93% of the sampled actions, the LLM correctly identifies preconditions that exactly match the ground truth, and for 86% of the actions, the extracted effects match the ground truth. We further analyze cases where the LLM outputs deviate from human annotations and identify the main sources of errors. First, some errors arise from application-specific dependencies between actions that are not explicitly defined in AALM. For example, certain Meterpreter commands such as Drop Token require a prior Steal Token action, and KeyStrokeDump requires KeyStrokeCapture to be enabled beforehand. Since these dependencies are tool-specific and not encoded in the current AALM predicate set, the LLM cannot reliably infer the correct preconditions and effects. Given the diversity of tools and applications involved in real-world attacks, it is impractical for AALM to predefine predicates for every specific application. Nevertheless, AALM is designed to be extensible: users can introduce new predicates to capture such dependencies, enabling the LLM to recognize and formalize these actions accordingly. Second, inaccuracies occur when a single complex action spans multiple tactics and produces multiple effects. In such cases, the LLM may fail to capture all effects accurately. This limitation is not unique to AALM or LLM-based extraction; similar challenges arise when modeling attack processes using MITRE TTPs, where a single procedure may serve multiple tactical purposes. Importantly, AURORA conservatively excludes actions whose predicates cannot be confidently formalized, preventing such inaccuracies from propagating into downstream symbolic planning.

We further evaluate the impact of the prompt engineering. In AURORA, we design tactic-specific prompts that explicitly guide the LLM through a reasoning process for identifying preconditions and effects. As an ablation baseline, we remove the prompt and instead provide the LLM with the complete set of predicates in AALM, allowing it to independently decide which predicates apply to a given action. The results show that, compared to the prompt-engineered version, the ablated setting produces inaccurate preconditions and effects for 58% of the evaluated actions, indicating that the LLM fails to exhaustively identify relevant conditions or effects when not explicitly guided by the prompt.

6 Limitation and Future Work

More Advanced Attack Linking with LLMs. We leveraged LLMs to analyze the preconditions and effects of attack actions based on raw documentation and AALM. We applied prompt engineering, injecting the step-by-step thinking process, illustrative examples, and candidate predicates into the prompt. A key future research direction is enabling LLMs to link actions automatically.

Apply AALM in Automated Penetration Testing. AALM links attack actions to enable symbolic planning for attack automation; the tools, actions, and predicates in AURORA are reusable for pentesting. However, to realize automated penetration testing with AALM, several additional challenges must be addressed, such as incomplete environment knowledge, non-deterministic effects, unmodeled predicates, and the need to convert observed information into PDDL format.

7 Conclusion

In this paper, we define attack actions and the attack action linking model to integrate heterogeneous attack tools into causality-preserved cyberattacks. We then introduce AURORA, the first automated, customizable, and causality-preserving cyberattack emulation system with LLM-powered symbolic planning. The evaluation demonstrates that AURORA can emulate customized, causality-preserved attack chains with high scalability and diversity.

8 Ethics Considerations

We take the ethical issues in attack emulation seriously. The attack tool analyzer and attack report analyzer can only analyze existing attack tools and reports without involving any zero-day attacks developed. Our goal is to leverage existing public knowledge of cyberattacks to enhance defensive capabilities.

Acknowledgments. We would like to thank anonymous reviewers for their constructive feedback. This project was supported by the National Science Foundation (NSF) grant 2148177 and funds from the Resilient & Intelligent NextG Systems (RINGS) program.

Disclosure of Interests. The authors declare that they have no competing interests.

References

1. Alam, M.T., Bhusal, D., Park, Y., Rastogi, N.: Looking beyond iocs: Automatically extracting attack patterns from external cti. In: Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses. pp. 92–108 (2023)
2. Alsaheel, A., Nan, Y., Ma, S., Yu, L., Walkup, G., Celik, Z.B., Zhang, X., Xu, D.: {ATLAS}: A sequence-based learning approach for attack investigation. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 3005–3022 (2021)
3. Applebaum, A., Miller, D., Strom, B., Korban, C., Wolf, R.: Intelligent, automated red team emulation. In: Proceedings of the 32nd annual conference on computer security applications. pp. 363–373 (2016)
4. of Basel, A.I.G.U.: Fast downwards (2024), `https://github.com/aibasel/downward`, accessed: 2024-11-01
5. BishopFox: Sliver (2024), `https://github.com/BishopFox/sliver`, accessed: 2025-04-01
6. Mitre caldera. `https://github.com/mitre/caldera`

7. Chen, G., Yang, L., Jia, R., Hu, Z., Chen, Y., Zhang, W., Wang, W., Pan, J.: Language-augmented symbolic planner for open-world task planning. arXiv preprint arXiv:2407.09792 (2024)
8. Chen, J., Hu, S., Zheng, H., Xing, C., Zhang, G.: Gail-pt: An intelligent penetration testing framework with generative adversarial imitation learning. Computers & Security **126**, 103055 (2023)
9. Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., Han, X.: Kairos:: Practical intrusion detection and investigation using whole-system provenance. arXiv preprint arXiv:2308.05034 (2023)
10. Cheng, Z., Lv, Q., Liang, J., Wang, Y., Sun, D., Pasquier, T., Han, X.: Kairos: Practical intrusion detection and investigation using whole-system provenance. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3533–3551. IEEE (2024)
11. Choi, S., Yun, J.H., Min, B.G.: Probabilistic attack sequence generation and execution based on mitre att&ck for ics datasets. In: Cyber Security Experimentation and Test Workshop. pp. 41–48 (2021)
12. CISA: Cybersecurity alerts advisories (2024), `https://www.cisa.gov/news-events`, accessed: 2024-07-01
13. Corporation, T.M.: Cve program mission. `https://www.cve.org/`
14. Official website of cymulate. `https://cymulate.com/`
15. Darpa engagement data. `https://drive.google.com/drive/folders/1okt4AYElyBohW4XiOBqmsvjwXsnUjLVf`
16. Database, N.V.: Official common platform enumeration (cpe) dictionary. `https://nvd.nist.gov/products/cpe`
17. Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Li, Y., Xu, Y., Zhang, T., Liu, Y., Pinzger, M., Rass, S.: Pentestgpt: An llm-empowered automatic penetration testing tool. arXiv preprint arXiv:2308.06782 (2023)
18. Ding, Y., Zhang, X., Amiri, S., Cao, N., Yang, H., Kaminski, A., Esselink, C., Zhang, S.: Integrating action knowledge and llms for task planning and situation handling in open worlds. Autonomous Robots **47**(8), 981–997 (2023)
19. Engenuity, M.: Cybersecurity: Att&ck® evaluations (2024), `https://evals.mitre.org/`, accessed: 2025-06-01
20. Evaluations, A.: Machine-readable fin7 emulation plan. `https://github.com/attackevals/ael/tree/main/Enterprise/fin7/Emulation_Plan/yaml`
21. Fortra: Software for adversary simulations and red team operations (2024), `https://www.cobaltstrike.com/`, accessed: 2025-04-01
22. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the american statistical association **32**(200), 675–701 (1937)
23. FunnyWolf: Viper (2024), `https://github.com/FunnyWolf/Viper?tab=readme-ov-file`, accessed: 2024-11-01
24. Ghanem, M.C., Chen, T.M., Nepomuceno, E.G.: Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks. Journal of Intelligent Information Systems **60**(2), 281–303 (2023)
25. Goyal, A., Wang, G., Bates, A.: R-caid: Embedding root cause analysis within provenance-based intrusion detection. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3515–3532. IEEE (2024)
26. Özeren Hacıoğlu, S.: Silver fox apt targets public sector via trojanized medical software. `https://www.picussecurity.com/resource/blog/silver-fox-apt-targets-public-sector-via-trojanized-medical-software`

27. Han, X., Pasquier, T., Bates, A., Mickens, J., Seltzer, M.: Unicorn: Runtime provenance-based detector for advanced persistent threats. arXiv preprint arXiv:2001.01525 (2020)
28. Hassan, W.U., Guo, S., Li, D., Chen, Z., Jee, K., Li, Z., Bates, A.: Nodoze: Combatting threat alert fatigue with automated provenance triage. In: network and distributed systems security symposium (2019)
29. Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., Steinhardt, J.: Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300 (2020)
30. Hoffmann, J.: Simulated penetration testing: From" dijkstra" to" turing test++". In: Proceedings of the international conference on automated planning and scheduling. vol. 25, pp. 364–372 (2015)
31. Holm, H.: Lore a red team emulation tool. IEEE Transactions on Dependable and Secure Computing **20**(2), 1596–1608 (2022)
32. Holm, H., Sommestad, T.: Sved: Scanning, vulnerabilities, exploits and detection. In: MILCOM 2016-2016 IEEE Military Communications Conference. pp. 976–981. IEEE (2016)
33. Holm, S.: A simple sequentially rejective multiple test procedure. Scandinavian journal of statistics pp. 65–70 (1979)
34. Are you vulnerable or exploitable? official website of horizon3. `https://horizon3.ai/`
35. Hossain, M.N., Sheikhi, S., Sekar, R.: Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In: IEEE Symposium on Security and Privacy (SP) (2020)
36. Hu, Z., Beuran, R., Tan, Y.: Automated penetration testing using deep reinforcement learning. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 2–10. IEEE (2020)
37. Husari, G., Al-Shaer, E., Ahmed, M., Chu, B., Niu, X.: Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources. In: Proceedings of the 33rd annual computer security applications conference. pp. 103–115 (2017)
38. Jia, Z., Xiong, Y., Nan, Y., Zhang, Y., Zhao, J., Wen, M.: {MAGIC}: Detecting advanced persistent threats via masked graph representation learning. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5197–5214 (2024)
39. Jiang, B., Bilot, T., El Madhoun, N., Al Agha, K., Zouaoui, A., Iqbal, S., Han, X., Pasquier, T.: ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems. In: Security Symposium (USENIX Sec'25). USENIX (2025)
40. Kumarasinghe, U., Lekssays, A., Sencar, H.T., Boughorbel, S., Elvitigala, C., Nakov, P.: Semantic ranking for automated adversarial technique annotation in security text. In: Proceedings of the 19th ACM Asia Conference on Computer and Communications Security. pp. 49–62 (2024)
41. Leigh, C.: Silver fox apt attack taiwan. `https://westoahu.hawaii.edu/cyber/global-weekly-exec-summary/silver-fox-apt-attack-taiwan/`
42. Li, Q., Wang, R., Li, D., Shi, F., Zhang, M., Chattopadhyay, A.: Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning. IEEE Transactions on Information Forensics and Security (2024)
43. Li, S., Dong, F., Xiao, X., Wang, H., Shao, F., Chen, J., Guo, Y., Chen, X., Li, D.: Nodlink: An online system for fine-grained apt attack detection and investigation. arXiv preprint arXiv:2311.02331 (2023)

44. Li, Z., Zeng, J., Chen, Y., Liang, Z.: Attackg: Constructing technique knowledge graph from cyber threat intelligence reports. In: European Symposium on Research in Computer Security. pp. 589–609. Springer (2022)
45. Lindemulder, G.: What are advanced persistent threats? https://www.ibm.com/think/topics/advanced-persistent-threats?regionCode=US&languageCode=en&contactmodule=true&cm-history=US-en
46. Loevenich, J.F., Adler, E., Hürten, T., Spelter, F., Roncevic, D., Lopes, R.R.F.: Automating cyber threat intelligence and attack chain generation using cyber security knowledge graphs and large language models. In: 2025 International Conference on Military Communication and Information Systems (ICMCIS). pp. 1–10. IEEE (2025)
47. Loevenich, J.F., Lopes, R.R.F.: Agentic generative ai for automation of cyber security attack chains in tactical manets. In: 2025 IEEE 50th Conference on Local Computer Networks (LCN). pp. 1–7. IEEE (2025)
48. Living off the land binaries, scripts and libraries https://lolbas-project.github.io/., https://lolbas-project.github.io/
49. Auromated breach and attack simulation market. https://www.marketsandmarkets.com/Market-Reports/automated-breach-attack-simulation-market-43164821.html
50. Milajerdi, S.M., Eshete, B., Gjomemo, R., Venkatakrishnan, V.: Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 1795–1812 (2019)
51. Milajerdi, S.M., Gjomemo, R., Eshete, B., Sekar, R., Venkatakrishnan, V.: Holmes: Real-time apt detection through correlation of suspicious information flows. In: IEEE Symposium on Security and Privacy (SP) (2019). https://doi.org/10.1109/SP.2019.00026
52. Miller, D., Alford, R., Applebaum, A., Foster, H., Little, C., Strom, B.: Automated adversary emulation: A case for planning and acting with unknowns. Tech. rep., MITRE CORP MCLEAN VA MCLEAN (2018)
53. Turla (2023) overview. https://attackevals.mitre-engenuity.org/enterprise/turla/
54. Att&ck matrix for enterprise. https://attack.mitre.org/
55. Ttp-based hunting. https://www.mitre.org/sites/default/files/2021-11/prs-19-3892-ttp-based-hunting.pdf
56. Msfvenom - metasploit unleashed. https://www.offsec.com/metasploit-unleashed/msfvenom/
57. Network, P.: What is an advanced persistent threat? https://www.paloaltonetworks.com/cyberpedia/what-is-advanced-persistent-threat-apt?utm_source=chatgpt.com
58. Atomic red team. https://github.com/redcanaryco/atomic-red-team, https://github.com/redcanaryco/atomic-red-team
59. Online: Explore atomic red team. https://atomicredteam.io/ (2025)
60. Online: Metasploit: The world's most used penetration testing framework. https://www.metasploit.com/ (2025)
61. OpenScap: Open source security compliance solution. https://www.open-scap.org/
62. OpenVAS: Openvas - open vulnerability assessment scanner. https://www.openvas.org
63. osquery: osquery: a sql powered operating system instrumentation, monitoring, and analytics framework. https://github.com/osquery/osquery

64. Pasquale, G.D., Grishchenko, I., Iesari, R., Pizarro, G., Cavallaro, L., Kruegel, C., Vigna, G.: ChainReactor: Automated privilege escalation chain discovery via AI planning. In: 33rd USENIX Security Symposium (USENIX Security 24). pp. 5913–5929. USENIX Association, Philadelphia, PA (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/de-pasquale
65. Welcome to pentera: Don't assume. validate. because 'pretty certain' doesn't mean secure. https://pentera.io/
66. Phillips, C., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: Proceedings of the 1998 workshop on New security paradigms. pp. 71–79 (1998)
67. Purple sharp. https://detectionlab.network/usage/purplesharp/
68. Rahman, M.R., Hezaveh, R.M., Williams, L.: What are the attackers doing now? automating cyberthreat intelligence extraction from text on pace with the changing threat landscape: A survey. ACM Computing Surveys **55**(12), 1–36 (2023)
69. Rapid7: Metasploit documentation of meterpreter (2024), https://docs.metasploit.com/docs/using-metasploit/advanced/meterpreter/meterpreter.html, accessed: 2025-04-01
70. Rehman, M.U., Ahmadi, H., Hassan, W.U.: Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 139–139. IEEE Computer Society (2024)
71. S3N4T0R-0X0: Apt attack simulation (2024), https://github.com/S3N4T0R-0X0/APT-Attack-Simulation, accessed: 2025-03-01
72. Sarraute, C., Buffet, O., Hoffmann, J.: Pomdps make better hackers: Accounting for uncertainty in penetration testing. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 26, pp. 1816–1824 (2012)
73. Satvat, K., Gjomemo, R., Venkatakrishnan, V.: Extractor: Extracting attack behavior from threat reports. In: 2021 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 598–615 (2021). https://doi.org/10.1109/EuroSP51992.2021.00046
74. Streamspot data. https://github.com/sbustreamspot/sbustreamspot-data
75. Symantec: Symantec enterprise blogs threat intelligence (2024), https://symantec-enterprise-blogs.security.com/threat-intelligence/clasiopa-materials-research, accessed: 2024-07-01
76. Takahashi, Y., Shima, S., Tanabe, R., Yoshioka, K.: {APTGen}: An approach towards generating practical dataset labelled with targeted attack sequences. In: 13th USENIX Workshop on Cyber Security Experimentation and Test (CSET 20) (2020)
77. Team, S.T.R.: Splunk attack range (2024), https://github.com/splunk/attack_range, accessed: 2024-07-01
78. Tiber-eu framework: How to implement the european framework for threat intelligence-based ethical red teaming. https://www.ecb.europa.eu/pub/pdf/other/ecb.tiber_eu_framework_2025~b32eff9a10.en.pdf
79. Wang, L., Shen, X., Li, W., Li, Z., Sekar, R., Liu, H., Chen, Y.: Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection. arXiv preprint arXiv:2404.14720 (2024)
80. 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. In: NDSS (2020)
81. 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. In: NDSS (2020)

82. 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 **35**, 24824–24837 (2022)
83. Wilcoxon, F.: Individual comparisons by ranking methods. In: Breakthroughs in statistics: Methodology and distribution, pp. 196–202. Springer (1992)
84. 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 cyber-attacks. arXiv preprint arXiv:2403.01038 (2024)
85. Yang, F., Xu, J., Xiong, C., Li, Z., Zhang, K.: {PROGRAPHER}: An anomaly detection system based on provenance graph embedding. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 4355–4372 (2023)
86. Zengy, J., Wang, X., Liu, J., Chen, Y., Liang, Z., Chua, T.S., Chua, Z.L.: Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In: 2022 IEEE symposium on security and privacy (SP). pp. 489–506. IEEE (2022)

A Appendix

A.1 Searching for Multiple Attack Chains using Symbolic Planners

To find as many attack chains as possible, we leverage the cost mechanism in symbolic planning. Initially, all actions have a cost of zero, and the algorithm searches for the plan with the lowest total cost in this action space (since all plans have a total cost of 0 at this stage, the search returns the first attack plan it discovers). After discovering a plan, we increase the costs of all actions within that plan by a fixed amount and update the domain. Then we search for the lowest-cost plan in the updated domain. Because the actions from the first plan now have higher costs, the planner favors creating new plans using previously unused actions. This iterative process continues until either no new plans are generated for several consecutive rounds or we reach our target number of generated plans.

A.2 Supplementary Experimental Results

Costs of Aurora: We assessed the time and monetary costs of Aurora. The results are shown in Table 7. With LLMs, extracting attack actions from documentation and analyzing their preconditions and effects takes less than 10 seconds on average. The time required to analyze one CTI report for attack customization is approximately 40 seconds. Planning time varies with the length of the attack chain, taking approximately 0.5 to 2 hours for a 50-step chain. Executing an emulated attack takes ten minutes on average. We also evaluate the cost of using LLM. The result shows that Aurora is economical in terms of both time and money.

Extracting TTPs from CTI Reports: We evaluate the accuracy of extracting TTPs from CTI using reports from CISA [12] as the ground truth. We employ LADDER [1] as the baseline. As shown in Table 8, the LLM-based report analyzer of Aurora achieves higher precision than the baseline, nearing or exceeding 80% for almost all reports. And the average recall is 69%. The

Table 7. Average Time and Monetary Costs of AURORA

	Time	LLM Costs	
		Tokens	Cost (USD)
Attack Tool Analyzing [1]	9.8s	10.8K	0.029
CTI Report Analyzing [2]	26.1s	16.8K	0.055
Reward Embedding [2]	14.9s	95.945K	0.010
Attack Planning [3]	0.5-2h	-	-
Attack Execution [3]	9.3m	-	-

[1] per attack action. [2] per report. [3] per attack chain (50 actions on avg.).

Table 8. The Performance Comparison of Attack Technique Extraction between LADDER and AURORA

id	Recall		Precision		Correctness	
	Base	Ours	Base	Ours	Base	Ours
1	61.5%	**100.0%**	42.1%	**100.0%**	52.6%	**84.6%**
2	28.6%	**71.4%**	17.4%	**100.0%**	30.4%	**100.0%**
3	20.0%	**100.0%**	5.6%	**28.6%**	22.2%	**80.0%**
4	28.9%	**88.1%**	43.3%	**97.2%**	43.3%	**100.0%**
5	33.3%	**93.3%**	41.7%	**100.0%**	46.2%	**100.0%**
6	58.3%	**62.5%**	35.9%	**93.8%**	43.6%	**100.0%**
7	17.1%	**38.3%**	25.0%	**78.6%**	42.9%	**80.0%**
8	19.0%	**76.2%**	25.0%	**88.9%**	37.5%	**94.4%**
9	25.0%	**100.0%**	11.8%	**53.8%**	35.3%	**84.6%**
10	46.2%	**84.6%**	22.2%	**84.6%**	33.3%	**100.0%**

result also shows that AURORA provides a more accurate description for each individual technique than LADDER. After examining the results, we identified two key advantages of using LLM. First, LLMs have better capabilities in understanding long sentences. Conversely, LADDER tends to be distracted by irrelevant words in filenames or positions. Second, LLM provides more comprehensible descriptions for the extracted TTPs. In contrast, LADDER sometimes yields incomplete sentences.

We also analyze the reasons for failure cases. First, LLM makes mistakes when some descriptions are ambiguous. For instance, a report mentions that *affiliates communicate with victims via TOR, Tox, email, or encrypted applications*. However, the LLM incorrectly identified this sentence as evidence of a command and control technique during the attack. Second, LLMs sometimes fail to enumerate all TTPs exhaustively. For instance, a report mentions that the attacker *impersonates company IT and/or helpdesk staff to gain trust and obtain credentials*. While the LLM successfully identifies Impersonation (T1656), it fails to recognize that this also demonstrates the attacker is gathering victim identity information (T1589).