

From Sands to Mansions: Towards Automated Cyberattack Emulation with Classical Planning and Large Language Models

Lingzhi Wang*, Zhenyuan Li[†], Yi Jiang[†], Zhengkai Wang[†], Zonghan Guo[†], Jiahui Wang[†]
 Yangyang Wei[†], Xiangmin Shen*, Wei Ruan[†], and Yan Chen*

*Northwestern University, [†]Zhejiang University

Abstract—As attackers continually advance their tools, skills, and techniques during cyberattacks - particularly in modern Advanced Persistence Threats (APT) campaigns - there is a pressing need for a comprehensive and up-to-date cyberattack dataset to support threat-informed defense and enable benchmarking of defense systems in both academia and commercial solutions. However, there is a noticeable scarcity of cyberattack datasets: recent academic studies continue to rely on outdated benchmarks, while cyberattack emulation in industry remains limited due to the significant human effort and expertise required. Creating datasets by emulating advanced cyberattacks presents several challenges, such as limited coverage of attack techniques, the complexity of chaining multiple attack steps, and the difficulty of realistically mimicking actual threat groups.

In this paper, we introduce modularized *Attack Action* and *Attack Action Linking Model* as a structured way to organizing and chaining individual attack steps into multi-step cyberattacks. Building on this, we propose AURORA, a system that autonomously emulates cyberattacks using third-party attack tools and threat intelligence reports with the help of classical planning and large language models. AURORA can automatically generate detailed attack plans, set up emulation environments, and semi-automatically execute the attacks. We utilize AURORA to create a dataset containing over 1,000 attack chains. To our best knowledge, AURORA is the only system capable of automatically constructing such a large-scale cyberattack dataset with corresponding attack execution scripts and environments. Our evaluation further demonstrates that AURORA outperforms the previous similar work and even the most advanced generative AI models in cyberattack emulation. To support further research, we published the cyberattack dataset and will publish the source code of AURORA in the future.

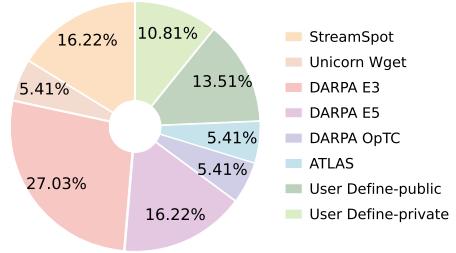
1. Introduction

As the technologies employed by attackers in cyberspace continually evolve, a comprehensive and up-to-date cyberattack dataset becomes essential for benchmarking the performance of defense systems and identifying their weaknesses. Besides, a dataset containing advanced cyberattacks is also beneficial for threat-informed defense [1], where attack data is used to train the defense models [2], [3]. Unlike the

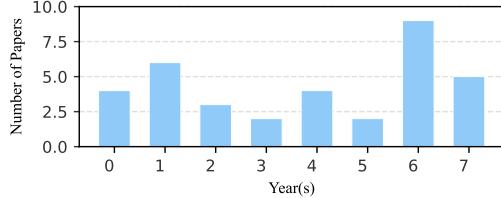
benchmark datasets for stable tasks like image classification and math problem solving, the datasets of cyberattacks need to be consistently updated due to the confrontational and rapidly evolving nature of cyberattacks. However, we observed a dataset drought in this area [1], [4], especially in terms of advanced persistent threats (APTs). Many recent academic studies on intrusion detection and mitigation rely on obsolete datasets. These datasets, which have not been updated for more than five years, limit the research progress toward more advanced attacks and probably produce biased evaluation results.

We reviewed the benchmark datasets used in recent papers [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19] on intrusion detection published at top-tier academic conferences. Figure 1a presents the distribution of the datasets; Figure 1b illustrates the time gap between the publication of the paper and the release of the dataset. From these two figures, we can learn that: 1. The benchmark datasets for intrusion detection systems in academia have been relatively homogeneous, primarily focused on a few public datasets such as DARPA Engagements [20], StreamSpot [21], and ATLAS [22]. Only 13.51% of them employed user-defined public datasets, indicating a lack of diversity in attack scenarios. 2. The time when a detection system is proposed is significantly later than when the benchmark dataset is published (with an average delay of 4.75 years), suggesting that many of the attacks may already be outdated by the time they are tested. The homogeneous and obsolete datasets raise concerns about biased results and saturated baselines.

We summarize three requirements for ideal cyberattack datasets, namely, *Reproducibility*, *Diversity*, and *Reality*. **Reproducibility** refers to the capability of a cyberattack dataset that can be replicated in real-world environments. Defense systems usually take data from various layers, such as network traffic, authentication events, and host-based system logs. This requires us to not only collect data but also providing details on how to replay the attacks, including the explicit, step-by-step instructions and the necessary infrastructures and environment to execute them so that the users can collect different data according to their needs. **Diversity** means that the cyberattack dataset needs to cover a wide range of attack surfaces, techniques, and tools. **Reality** means that the generated attack dataset should reveal the preferences and behaviors of specific real-world attackers.



(a) Proportional use of cyberattack datasets in recently published intrusion detection papers.



(b) Time lags between the publication of paper and publication of cyberattack datasets used in the paper (Years).

Figure 1: Review of dataset used in intrusion detection papers

Attack technique libraries, such as Metasploit [23] and Atomic Red Team [24], provide individual implementations of some attack techniques, which can be utilized to test defense systems or create datasets. However, they are unable to produce complete, multi-step attack chains. Modern cyberattacks, such as APT attacks, typically involve multiple stages ranging from initial reconnaissance and access to data exfiltration [25] and span for a long time. And many intrusion detection approaches [16], [26] also rely on correlating different stages during the long-term attack, thus it is improper to evaluate these systems with isolated attack steps. Tools like Caldera [27], PurpleSharp [28], and Attack Range [29] enable the execution of multi-step playbooks. However, these playbooks are typically written manually by security experts.

We summarized the primary challenges in constructing a multi-step cyberattack dataset with reproducibility, diversity, and reality. The first challenge is the limited, decentralized, and unstructured knowledge about attack techniques. A diverse cyberattack dataset must include a wide range of Tactics, Techniques, and Procedures (TTPs). Yet existing approaches use only a limited set of TTPs when constructing attacks. For example, the state-of-the-art automated attack planner, ChainReactor [30], focuses on privilege escalation attacks and only considers around 30 available attack actions during planning. Integrating attack techniques from diverse sources into attack emulation requires extensive human effort and expertise.

The second challenge is constructing multi-step cyberattacks. In recent years, many studies [30], [31], [32] have focused on automating multi-step planning for cyberattacks using classical planning techniques, such as the Planning Domain Definition Language (PDDL). Generating multi-

step attack plans requires understanding the relationship between attack steps. However, in cyberattacks, these relationships are complicated. Therefore, existing works cannot propose a framework to model and describe the relationship between attack steps. The third challenge is aligning attack emulation with Cyber Threat Intelligence (CTI) reports. This is crucial for creating realistic datasets - we want the constructed cyberattacks to resemble the behaviors of real attack groups documented in CTI reports. However, despite advances in CTI analysis, how to align attack generation with human-written reports remains an unanswered question.

To address those challenges, we propose AURORA, an automated cyberattack emulation system. AURORA incorporates third-party attack tools in attack emulation. By converting the attack techniques in documentation to modularized *Attack Actions* and connecting them using *Attack Action Linking Model*, AURORA can apply classical planning algorithms to generate multi-step attack chains. AURORA can mimic the behaviors of specific attack groups reported in CTI reports. To generate fully reproducible datasets, AURORA provides attack execution scripts in Python and attack emulation environments for each generated attack. We identify the capability of Large Language Models (LLMs) in cyberattack emulation. First, LLMs can effectively analyze unstructured attack tool documentation to help extract attack actions. Second, the generative ability of LLMs helps describe unseen relationships between attack actions. Lastly, LLMs can help analyze human-written CTI reports.

We evaluate AURORA using over 50 CTI reports and four widely used third-party attack tools. AURORA extracts more than 5,500 attack actions from these tools. Using these actions, we constructed over 5,000 attack chains. After manually validating over 1,000 of these chains, we published them as a cyberattack dataset (which continues to grow). Other experiment results demonstrate the superior capability of AURORA compared to the best existing automated attack emulation framework and most advanced generative AI.

In this paper, we make the following contributions:

- C1. We propose AURORA, an automated cyberattack emulation system, which integrates third-party attack tools and connects them into multi-step attacks. It can analyze CTI reports and replicate attack patterns from real-world threat groups. AURORA sets up attack simulation environments and generates semi-automated Python scripts to execute attacks.
- C2. We defined *Attack Actions* to unify and modularize various attack tools and *Attack Action Linking Model* to connect them. The combination of classical planning and LLMs addresses the challenges in cyberattack construction, such as the lack of available attack tools and manual effort in multi-step planning. It also enables us to emulate attackers according to CTI reports.
- C3. We conducted extensive experiments on AURORA. The evaluation shows that AURORA can effectively connect more diverse attack actions into attack chains with higher quality compared to baselines, including the most advanced generative AI.

- C4. We constructed, manually verified, and published over 1,000 attack chains. To our knowledge, this is the first automatically generated and the largest cyberattack dataset to date. We released the dataset to the public for further research¹ and will release our system in the future.

2. Background and Related Work

2.1. Automated Cyberattack Dataset Generation

The lack of an open, comprehensive, and updated cyberattack dataset has been recognized by more and more researchers [1], [4]. In [1], Choi et al. employ the hidden Markov model (HMM) to generate attack sequences. Takahashi et al. proposed APTGen [4], a system that can generate attack sequences from the CTI reports, execute them, and collect the corresponding traces. However, the generated attack sequences from these approaches do not contain details for automated execution. The attack steps in the sequence should be implemented and executed manually.

Some researches [2], [3], [33], [34], [35], [36] attempt to construct knowledge databases about cyberattacks by dissecting CTI reports to profile attacker behaviors. However, they primarily focus on high-level MITRE tactics and techniques since attack reports typically provide broad descriptions rather than specific execution details such as tools, scripts, and commands. For example, the system may extract an “Execution via DLL Injection” technique from the reports but provide no executable implementation for this technique. Moreover, many CTI reports only share knowledge about critical attack steps rather than complete attack chains. For instance, reports often describe a malware’s post-exploitation activities without explaining how the malware initially infected the victim host. These gaps make it hard to fully reproduce the attacks in real systems.

Another related research direction is automated penetration testing [37], [38], [39], [40] and red teaming [25], [41]. Penetration testing focuses on identifying and exploiting vulnerabilities on a target network, while red teams design more complete attack plans, including both vulnerability exploitation and post-exploitation actions. However, these approaches focus on creating one single attack chain against a given target, thus failing to build comprehensive datasets containing various attack scenarios. They are also unable to mimic real-world attacker behaviors described in CTI reports. Additionally, existing cyberattack automation systems, particularly those model cyberattacks as logical and conceptual steps (e.g. Attack Graphs [42], [43], [44]), also suffer from the lack of executing details.

Recent advancements in LLM are also reshaping cyberattack automation. Research by Happy et al. [45] demonstrated how LLMs can be used for high-level task planning in penetration testing and vulnerability scanning. Other studies [46] have explored LLMs for automating specific

tasks, such as privilege escalation on Linux systems. Additionally, works by [25], [40] have focused on developing LLM-based modules for high-level decision-making, action generation, and result analysis, with a particular emphasis on sophisticated attack construction using tools. Although LLMs can provide specific instructions and commands for attack execution, it is hard for them to link different attack steps and construct multi-step cyberattacks, which we will discuss later in §2.3.

2.2. Classical Planning and PDDL

AURORA employs *classical planning* (also known as *symbolic planning*) to build multi-step cyberattacks automatically. In general, a planning problem aims to find a sequence of actions that achieve a specific goal [47], [48]. Classical planning employs *Planning Domain Definition Language (PDDL)*, a declarative language, to define planning problems with symbolic notations. In this section, we introduce the basic backgrounds about classical planning and PDDL. For more details, we recommend reading related papers [47], [48], [49].

A classical (symbolic) planning problem is usually 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 before the plan starts, and G is the goal state that the plan wants to achieve. An action $a \in \mathcal{A}$ is the basic unit that can change the states. An action includes two important features: the *preconditions*, which are the predicates that must be true or false to apply the action, and the *effects*, which are the predicates that will become true or false after the action. A plan is a sequence of actions $\{a_1, a_2, \dots, a_n\}$. Starting from I , the precondition of a_i is satisfied by the state s_{i-1} (the precondition of a_1 should be satisfied by I) and after executing a_i , the state is changed from s_{i-1} to s_i . After executing the last action a_n , we should arrive at the goal state G . PDDL is a standardized language to define those aforementioned concepts in planning problems.

```
(:action get_powershell_session_using_sliver
:parameters (?slv_id - id ?psh_id - id ?t - host)
:precondition
  (and
    (sliver_session ?slv_id ?t)
    (not (= ?slv_id ?psh_id)))
:effect
  (and
    (powershell_session ?psh_id ?t)
    (increase (total-reward) 2))
```

Figure 2: An illustrative example of an action in PDDL

Fig. 2 shows an example of an action defined in PDDL. It presents an action for obtaining a PowerShell session using Sliver, a popular C2 framework in cyberattacks. The precondition of this action is a predicate indicating that a

1. <https://github.com/LexusWang/Aurora-demos>

Sliver session must be established on the target host. The effect is a predicate showing that a PowerShell is obtained on the target host. Please note that preconditions and effects in PDDL support logical operations such as “and”, “or”, and “not”. Parameters are variables in the precondition and effect predicates. The action in this example has three parameters: `slv_id`, `psh_id`, and target host `t`. Additionally, each action can be assigned a numerical function (“total-reward” in the example) for numerical optimization. We will show the usage of them in §4.3.1.

A planning problem written in PDDL usually consists of two files: a domain file and a problem file. The domain file specifies all predicates \mathcal{P} , actions \mathcal{A} , and data types in the target domain. The problem file defines the starting state I , the goal state G , available objects, and the optimization metric for a specific problem. Due to space constraints, we present a detailed example of the problem file in Appendix §A. With the planning problem defined in these two files, domain-independent planning algorithms can search for the plan automatically.

2.3. Cyberattack as a Planning Problem

Framing cyberattacks as a planning problem is not a new topic, as many works have applied classical and AI planning methods to penetration testing and red team emulation. Starting from [42], [50], numerous works [39], [51], [52], [53], [54], [55] have attempted to automate penetration testing by simplifying attack capabilities or the target networks as a graph. The cyberattacks are modeled as “hops” between different nodes [32]. However, these works mainly focus on vulnerabilities and exploits during planning, while, as shown by the MITRE ATT&CK matrix [56], modern cyberattacks encompass a larger action scope, including the post-exploitation actions.

Studies on red team emulation [41] try to address limitations by incorporating more post-exploitation actions. However, a fundamental question has not been clearly answered: how to define the relationship between attack actions. Specifically, how do we define actions, preconditions, and effects in cyberattacks? Penetration testing systems only consider basic conditions such as operating systems, vulnerabilities, exploits, and network topology. In CoreSec-POMDP [32], [57], states are described with a small set of predicates such as “connected”, “has *” and “compromised”, which cannot demonstrate the complex relationships between actions. In [31], [58], only three factors - accounts, hosts, and credentials - are considered to describe the attack process. Although tools like Caldera [27] support connecting actions by aligning their parameters, how to establish connections between parameters remains unanswered.

2.4. Assumptions

This paper aims to emulate cyberattacks to generate datasets and evaluate defense systems. This differs from penetration testing and red teaming, which simulate attackers working against unknown targets. In contrast, we

assume that all relevant information is known in advance, including the available attack techniques and environments. Our system does not involve reconnaissance of unknown targets or decision-making under incomplete information. For example, we do not guess whether a specific vulnerability exists on a target machine. Instead, if a generated attack involves such a vulnerability, we construct a corresponding vulnerable machine and execute the attack accordingly - while we can simulate the reconnaissance and guessing behavior of an attacker, all information is known before planning starts.

3. Attack Action Linking Model

Before introducing how we construct cyberattacks in detail, we first define *Attack Actions* and *Attack Action Linking Model*, which serve as the basis of our cyberattack construction system. In short, *Attack Actions* organize the tools, techniques, and actions of the attackers in a modularized and structured way. And the *Attack Action Linking Model* describes the relationship between Attack Actions to integrate them into a multi-step attack chain.

3.1. Attack Actions

In classical symbolic planning, “Action” is the smallest atomic unit forming a plan. Similarly, we define an *Attack Action* as an atomic operation that can be performed during cyberattacks.

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 content of an attack action depends on how different attack tools are executed and the desired level of granularity in cyberattack emulation. For example, in Atomic Red Team [24], each test can be regarded as an attack action. In Metasploit [23], each module serves as an attack action. In some post-exploit tools like Sliver [59] and Cobalt Strike [60], each command in the console is defined as an attack action. Users can bind multiple small actions to create a larger attack action. For example, in Metasploit, an exploit module can be packaged with a payload module to form a single attack action. Readers may associate the concept of attack action with MITRE ATT&CK procedures, which is also the smallest unit in the tactics, techniques, and procedures (TTPs) model. They both illustrate specific implementations in cyberattacks. The difference is that attack actions are more structured and contain features needed to organize and execute them in a cohesive, multi-step cyberattack.

Table 1 summarizes the features associated with each attack action. We also show an example of the attack action in Figure 12 in Appendix B. Specifically, *UUID*, *name*, *source*, and *description* specify the basic information of the attack action. *Supported platform* presents the operation systems that support executing this action. MITRE ATT&CK *tactics*

TABLE 1: Features of attack actions

Field	Description
UUID	A universally unique identifier.
Name	A short name for the attack action.
Description	A brief description of the attack action.
Source	Where is the action from?
Platform	Which OS is the action applicable to?
Tactics	MITRE ATT&CK tactics.
Technique	MITRE ATT&CK technique.
Execution	The concrete executor and command to execute.
Preconditions	Conditions that must be satisfied for action execution.
Effects	Conditions that will be satisfied after action execution.

and *techniques* are essential for aligning with CTI reports, which we will introduce in the following sections.

Execution field provides the detailed command or instructions to execute this action. It comes with the corresponding *Executor*. Each action, whether it involves a script, command, cmdlet, or exploitation, must be executed with some executors. For example, a malicious Powershell cmdlet requires the attacker to obtain an interactive Powershell or the capability of executing a Powershell script on the target hosts. Therefore, the executor can be defined as “powershell”. All commands provided by Meterpreter [61] require a live Meterpreter session established on the target, thus having “meterpreter” as the executor. Some action needs interaction from the victim, e.g. clicking the phishing link to download and execute a malware, which has “user” as the executor.

The *Precondition* and *Effect* features are used to link different actions. Therefore, we introduce them in the next section together with the attack action linking model.

3.2. Attack Action Linking Model

As introduced in §2.2, *preconditions* and *effects* are used to link actions together and form a plan in classical planning. Preconditions are the required conditions that must be met before executing an attack action. For example, in Figure 2, executing this action requires an active Sliver session on the target machine. Effects are the resulting conditions after executing an attack action. For example, executing the action in Figure 2 results in a PowerShell session. An action is included in an attack plan only if 1) all its preconditions are fulfilled, and 2) its effects help satisfy the preconditions of following actions or achieve specific attack goals.

However, “condition” is a broad, subjective, and ambiguous concept. For instance, if we simply ask “*What are the preconditions of uploading a file to the target machine using the “upload” command in the Meterpreter session?*”, different people will give different answers. A short answer could be “*An active Meterpreter session, a file to be uploaded, and permissions to create a new file.*” But strictly speaking, network accessibility, sufficient disk space on the target machine, correct file path, and avoiding detection can all be regarded as preconditions. However, during real-world attacks, attackers might not consider these trivial preconditions.

Therefore, a standard, universal, and succinct framework is needed for defining preconditions and effects of attack actions from the perspective of cyberattack emulation. In this paper, we propose *Attack Action Linking Model (AALM)* to describe the preconditions and effects of attack actions for linking them into attack chains.

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

In this paper, we design the predicate set of the attack action linking model from the following nine dimensions. Due to space limits, we cannot detail each predicate here; the complete model and predicates are presented in the Appendix B.

Environments: The first dimension is environments, which includes operating systems, vulnerabilities, software, and so on. The environments are crucial for cyberattack emulation. For instance, a shell obtained through a Linux vulnerability can only execute Linux-based commands. Similarly, an exploit targeting a specific CVE can only attack a host that contains that vulnerability. Representative predicates include (*OS_windows ?target-host*), (*CVE_exists ?target-host*), (*{Software}_running ?target-host*).

Executor: The second dimension is the executor. In §3.1, we explained what an executor is for an attack action. Executors play a crucial role in multi-stage cyberattacks: attackers typically aim to obtain an executor on the victim host (such as a Shell), which serves as the foundation for subsequent steps. Executor-related predicates fall into two groups: Type-related predicates, such as (*command_prompt ?executorID-executor ?target-host*), which means we have a command prompt shell on the target host; Privilege-related predicates such as (*elevated ?executorID-executor*), which indicate the executor has an elevated privilege.

Payloads: The third predicate category is related to payloads. Attackers generate payloads in different ways, which are executed by different executors. This category includes predicates describing payload types and payload operations. For example, (*sliver_implant ?p-payload*) indicates that the payload is a Sliver implant, while (*payload_executed ?p-payload ?target-host*) denotes the execution of the payload on the target host.

Files: The fourth dimension is the files, which are essential in multi-stage cyberattacks. For example, attackers must transfer generated payload files to the victim host, and when stealing sensitive data, they usually compress it into archives and send. File-related predicates are divided into two categories: file operations and file types. File operation predicates describe conditions about file existence, deletion, execution, and permission modifications. File type predicates identify the specific type of file, which is important when linking attack actions that involve file execution.

Processes: The fifth dimension is about processes, which includes the predicates regarding process status and process operations.

Users: The sixth dimension is users, encompassing both user types and operations performed on users, e.g. (`(root_user ?u-user)` and `(user_exists ?u-user ?target-host)`).

Credentials: The seventh category relates to credentials. We have observed that credentials often serve as the lynchpin connecting multiple attack stages - particularly those involving lateral movement, where an attacker must first obtain the username and password of the victim.

Information: The eighth dimension is information, which is any victim-related data other than credentials. Attackers can obtain information in different ways, and it may be leveraged in subsequent attack stages. Because the range of attack-related information is so broad, we have not defined this category as a fixed set; instead, information-related predicates typically comprise a large collection of predicates in the following form: `{info_detail}_info_known ?target - host`, where the `info_detail` depends on the specific information type.

Data: The last dimension captures the relationship between an attack actions in terms of data; these predicates typically appear in conjunction with file-related predicates. For example, `(Screenshot _data_saved ?f-file ?path-path ?target-host)` represents the effect of saving the screenshot to files on the victim host.

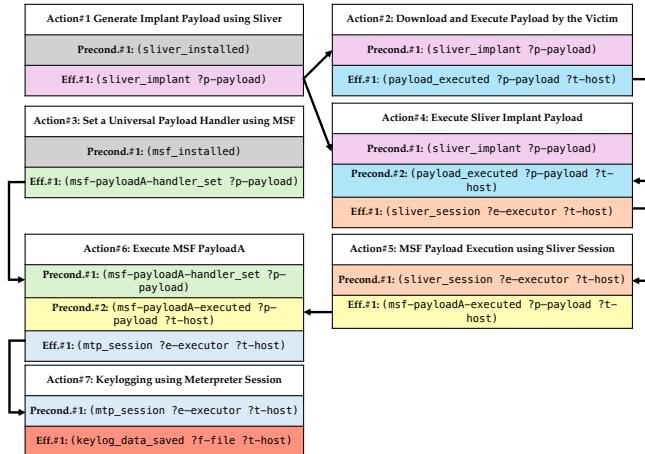


Figure 3: An example attack chain connected by predicates.

Figure 3 shows an example of an attack chain connected by predicates defined in AALM. We omitted some predicates due to space constraints. The corresponding explanations of each predicate can be found in Figure 13 in Appendix B.

We have to admit that the dimensions presented above cannot fully define and describe all relationships between attack actions (Indeed, given the complexity of cyberattacks, we believe a complete, one-and-done linking model is unattainable). However, as demonstrated in our experimental section, our Attack Action Linking Model is sufficient to

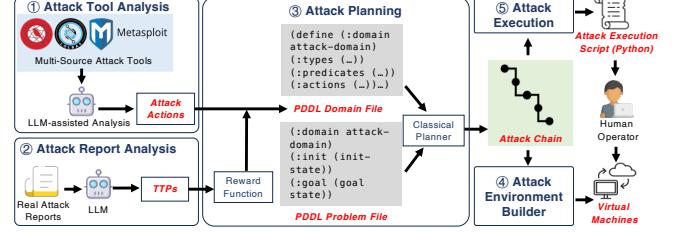


Figure 4: System overview of AURORA

effectively link different Attack Actions into an attack chain. Moreover, our Attack Action Linking Model is highly extensible, and as the first work to propose a standardized linking model to form attack chains covering both pre-exploitation and post-exploitation attacks, we hope to collaborate with more security practitioners to make the Attack Action Linking Model more comprehensive in the future. We discuss the limitations of AALM in the experimental and discussion sections.

4. System Overview

In this section, we introduce the design of AURORA, an automated cyberattack emulation system based on the attack actions and attack action linking model. As shown in Figure 4, AURORA takes the documentation of attack tools and CTI reports as input. It automatically outputs attack chains, sets up attack emulation environments, and generates scripts to execute the attacks semi-automatically in the environments. AURORA encompasses five components: ① Attack Tool Analyzing, ② Attack Report Analyzing, ③ Attack Planning, ④ Attack Environment Building, and ⑤ Attack Executing.

The attack tool analyzer analyzes documentation of attack tools, red team libraries, and penetration frameworks and converts them to attack actions. The attack report analyzer examines CTI reports to extract TTPs used by specific real-world attack groups. The attack planner converts the attack actions into the PDDL format and chains them with classical symbolic planning. It orchestrates the attack actions to attack chains, mimics real-world attack groups, and generates scripts to execute the attack chains. The attack environment builder automatically deploys the virtual machines needed to execute the generated attacks. Finally, the constructed cyberattacks are executed on the virtual machines semi-automatically with a Python script. In the following sections, we will detail the design of each component.

4.1. Attack Tool Analyzer

Even advanced cyberattack groups rarely build their attacks from scratch. Instead, they utilize third-party attack tools or maintain internal repositories of reusable toolkits. In this paper, we focus on building cyberattacks using existing attack tools and techniques rather than developing zero-day attack methods. This ensures our system can only be

used to evaluate and strengthen defense systems against known attack techniques (further discussion of ethical considerations can be found in §7). Additionally, by incorporating various tools used by real-world attackers, AURORA generates executable cyberattacks, integrates diverse attack techniques, and enhances attack fidelity. In contrast, many previous work [30], [31], [58] lack third-party attack tool integration, which limits their offensive capabilities. The attack tool analyzer of AURORA reviews the documentation of cyberattack tools, which is usually written in natural language, and converts them into structured attack actions defined in §3.

Some features of attack actions, such as name, description, source, and platform, are provided by most attack tools in their documentation. We utilize regular expressions to match and extract these features from the raw documentation texts. For those tools without MITRE ATT&CK labels in their documentation, we employ an LLM to analyze the documentation and infer the MITRE ATT&CK labels. Instead of relying on zero-shot inference (simply asking an LLM to identify MITRE ATT&CK labels from documentation), we explore how prompt engineering and fine-tuning could improve LLM labeling accuracy. For prompt engineering (PE), we use a two-step approach (Chain-of-Thought [62]). First, we ask the LLM to identify the MITRE tactic - a simpler task given the limited options. Then, based on this answer, we retrieve the relevant techniques within that tactic. These techniques are presented as options in a second prompt for the LLM to select from. The experiment results show that this approach significantly improves labeling accuracy and prevents hallucination issues. For fine-tuning (FT), we utilized text-label pairs from the MITRE website [56] to fine-tune the LLM. The experimental results indicate that fine-tuning actually decreased the labeling accuracy. We illustrate and analyze the results in §5.5.1.

The most challenging part of converting documentation to attack actions is assigning the precondition and effect predicates. The challenge is twofold: we must limit the variety of predicates to link related actions while also creating new predicates to accommodate unseen preconditions and effects. Firstly, the generated predicates must be standardized and limited in variety, as classical planners depend on same predicates to connect the preconditions and effects of attack actions. Therefore, we must ensure that related attack actions have consistent precondition and effect predicates and avoid generating excessively diverse predicates, which would prevent them from being connected. Second, the predicates must be extensible and adaptable. As mentioned in §3.2, it is impossible to define a complete set of predicates for all preconditions and effects. Therefore, the system must be able to generate new predicates to handle unseen preconditions and effects.

We design the precondition and effect analysis pipeline, as shown in Figure 5. For each attack action, we first assign precondition and effect predicates based on some rules. For example, we determine environment predicates based on the supported operating system and CVE list of a Metasploit exploit module. For Metasploit payload mod-

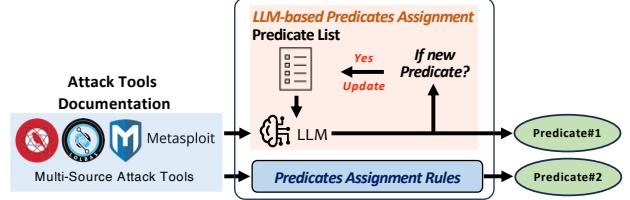


Figure 5: The pipeline of assigning precondition and effect predicates.

ules, the precondition predicates include compatible exploit modules listed in their documentation. Similarly, we assign a “sliver executor” predicate to all Sliver session commands as the precondition. We directly assign precondition and effect predicates according to rules designed based on our expertise on the attack tools.

However, there are still some attack actions whose preconditions and effects need deeper analysis, i.e. cannot be extracted using a rule. We use LLMs to analyze documentation and determine their precondition and effect predicates. To maximize connections between new actions and existing actions, we maintain a predicate list and require the LLM to first select from these existing predicates. Only when the LLM cannot find a suitable predicate in the given list will it generate a new predicate and add it to the predicate list for subsequent analysis. To help LLM better generate new predicates, we include a detailed AALM predicate format description in the prompt - what we call *guidance in prompt* (*GP*). We also include existing actions with their preconditions and effects, which we name as *examples in prompt* (*EP*). We evaluate the LLM’s accuracy in predicate assignment in §5.5.2.

4.2. Attack Report Analyzer

Real-world attack groups exhibit different preferences on the Tactics, Techniques, and Procedures (TTPs) they adopt. As mentioned in §1, AURORA aims to mimic different attack groups to generate realistic and diverse cyberattack datasets. To achieve this, AURORA leverages CTI reports, which document the behavior patterns of real attackers, as references. We design the reward function based on classical planning (details in §4.3.1) to generate attack chains that closely resemble CTI reports. The attack report analyzer of AURORA extracts TTPs from these reports.

In recent years, various approaches have been proposed to profile attackers by analyzing CTI reports using traditional Natural Language Processing (NLP) techniques [2], [3], [33], [34], [35] or LLMs [36]. In our experiments (§5.5.3), we found that the latest LLM surpasses these methods in terms of TTP extraction accuracy. Therefore, we use LLM to extract TTPs from the reports. To mitigate the hallucination issue, we include MITRE ATT&CK TTP definitions in the prompt. We also incorporate the output template into the prompts, as shown in Figure 14, to standardize the output in JSON format. The improvement of

CTI report analysis is not the focus of this paper and can be considered as a direction for future research.

4.3. Attack Planning

4.3.1. Generate PDDL Domain Files. Generating PDDL domain files seems to be straightforward since we have already generated all predicates and actions in §4.1. However, to mimic real attackers and build attacks that align with the TTPs extracted from reports, AURORA assigns a reward function to each attack action in the PDDL domain files. We reveal our preference for specific attack actions using the reward function r . Attack actions that appear in the report receive higher rewards, while others have lower rewards. In practice, we first determine if an action appears in the TTP set extracted from a report based on its MITRE label. If so, r is the cosine similarity between the embeddings of the action description and the description of the corresponding TTP. Therefore, even for attack actions with the same MITRE label, rewards vary according to the detailed descriptions. For example, consider two privilege escalation actions on Windows: bypassing User Account Control (UAC) and process injection. Both methods can elevate the attacker’s privilege, but if bypassing UAC is mentioned in the report, we prefer using it in our attack. In contrast, if process injection is not mentioned in the report, it retains a low reward. The planning algorithms attempt to maximize the rewards of the generated attack, making UAC bypass more likely to be selected.

Please note that necessary actions will still be chosen by the planner, even if they are not mentioned in the reports and thus have a zero reward. Random rewards can be assigned to actions to enhance the diversity of generated attack plans. We evaluate the effectiveness of the reward function in §5.4.

4.3.2. Generate PDDL Problem Files. PDDL problem files specify the starting state I and the goal state G of a planning problem. As mentioned in §2.2, the starting state I includes predicates set to true before the planning starts. In our experiments, we set environment predicates of all available operating systems, vulnerabilities, and software as the starting states. The goal state G specifies the desired ending predicates for an attack plan. In our experiments, we create a PDDL problem file for each predicate as the goal state to generate as many attack chains as possible. In real testing, the goal state can be customized as the combination of multiple predicates.

With PDDL domain files and problem files ready, various off-the-shelf planning algorithms can be applied to find attack chains. We design a method to search for as many attack chains as possible to maximize the diversity. The details can be found in Appendix D.1.

4.4. Attack Environment Builder

AURORA also aims to build attack emulation environments where we can execute generated attack chains. Some existing works [63], [64], [65] are capable of automatically

building vulnerable environments as the attack targets. However, they either build environments within containers [63] or only focus on Linux kernel vulnerabilities [65], which is unsuitable for our use case. First, we prefer to use virtual machines (VM) rather than containers for attack emulation since VM can provide more isolated virtualization; second, Linux kernel vulnerabilities are not commonly seen in APT attacks. In this paper, we use another method for quickly building attack environments. We observed that there are numerous virtual machines available online for educational and penetration testing purposes, which we considered a valuable resource. We collected these virtual machine images and cataloged their running operating systems, Common Vulnerabilities and Exposures (CVE), and third-party software, then generated corresponding environment predicates. These predicates are the initial states in PDDL problem files. During planning, attack chains are generated based on these predicates. We then analyze the environment predicates in the attack chains and automatically pull and deploy corresponding virtual machines. Our experiments utilize 14 different virtual machine images, which users can download and deploy with a single click in our cyberattack dataset.

While this helps us set up attack environments for generated attack chains, we have to admit that the number of available VMs is limited. In fact, automatically building vulnerable environments remains an open question. To our knowledge, there is no solution capable of automatically configuring any arbitrary CVE and software. Moreover, many attack scenarios rely on misconfigurations rather than vulnerabilities. Addressing this open question is beyond the scope of this paper. We discuss the challenges in detail in §6.3.

4.5. Attack Execution and Data Collection

The last step of AURORA is executing the generated attack in the built environments. Different from previous attack planning [30] and emulation work [31], [41], [58], AURORA provides exact commands for each attack action. Moreover, since AURORA leverages widely used third-party attack tools, the Python API for these tools makes it possible to execute attack chains simply by running a Python script. However, in real practice, some human intervention is unavoidable for parameter configuration and user interaction. This is why we classify AURORA as a semi-automated attack execution system. We discuss fully-automatic attack execution as one of the future work in §6.2 and evaluate the time used in execution in §5.6. Another benefit of emulating live attacks is that users can choose different data collectors to gather attack traces for testing. To demonstrate this, we collected some data and used it to test several advanced intrusion detection systems in §5.7.

5. Evaluation

In this section, we evaluate the performance of AURORA from different perspectives. The evaluation is pivoted on the following research questions (RQs):

- RQ1. Can AURORA generate attack chains effectively and correctly with the help of AALM?
- RQ2. Can AURORA build a diverse cyberattack dataset?
- RQ3. Can AURORA mimic the behaviors of real-world attackers documented in CTI reports?
- RQ4. Does LLM help cyberattack emulation? How accurate and reliable are the LLM-based components in AURORA?
- RQ5. What are the time and monetary costs of using AURORA?
- RQ6. Can cyberattacks generated by AURORA help evaluate detection systems?

5.1. Evaluation Setup

External Attack Tools. The attack tool analyzer of AURORA converts existing third-party attack tools to attack actions. In this paper, we utilize Atomic Red Team [66], exploit modules and payload modules from Metasploit [23], Meterpreter [61], and Sliver [59]. We believe they represent typical cyberattack tooling: Metasploit is the most widely used penetration testing framework for pre-exploitation tasks. Atomic Red Team offers the largest post-exploitation tool library, though it's mainly for research and testing purposes. Meanwhile, Meterpreter and Sliver represent mature tools that attackers use in real-world post-exploitation scenarios. Some attack libraries and tools were not included in this paper due to licensing costs [60] or lacking details for execution [67], [68]. AURORA can incorporate new tools, which we view as the application and extension of the system. The above attack tools provide AURORA with 5,555 attack actions - more than many other automated cyberattack system [30], [40].

CTI Reports. We collected more than 50 CTI reports from different sources [3], [69], [70], [71] for AURORA to mimic attack groups. We did not consider reports focused on individual vulnerabilities or single malware because we aim to imitate APT attack chains that involve multiple stages. The CTI report dataset used in our evaluation will be open-sourced in our code repository.

Implementation and Evaluation Setup. We implemented AURORA with 4K LoC in Python. In most experiments, we use the GPT-4o from OpenAI as the LLM model (we also include several other LLMs in some experiments for comparison) and LangChain as the framework to interact with LLM. We use `text-embedding-ada-002` for text embedding due to its efficiency. We employed `pddl` [72], an open-source PDDL parser, to generate, edit, and manage the PDDL files. AURORA employs Fast Downward [73] to solve classical planning problems. We set up the attack emulation environment infrastructure using Oracle VirtualBox, creating a small-scale LAN with 15 different hosts. We performed all experiments on an Ubuntu 22.04.3 Linux Server with an Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz and 1.0 TB memory. We will release the code, datasets, and results to the public to facilitate further research.

Baselines. Since no similar systems exist for direct comparison, we evaluate AURORA against multiple related works

as baselines. The baselines encompass four categories. The first category is the APT emulation datasets, including MITRE Evaluation [74], APT Attack Simulation [75], PurpleSharp [28], and Attack Range [29]. These datasets contain multi-step attack chains and detailed steps to reproduce these attacks. However, unlike AURORA, none of these attack chain is generated automatically. The second category of the baseline is the single-step attack simulation framework, including Atomic Red Team [24], Metasploit [23], and Viper [68]. The third category of the baselines is the academic studies [31], [41], [58] on cyberattack automation. However, most of them do not publish their systems. Additionally, these approaches fail to generate concrete, executable attack commands within their plans, making it challenging to implement them in real environments. Chain-Reactor [30] provides a well-organized open-source codebase. However, it primarily focuses on privilege escalation in penetration testing and lacks other attack techniques. The last category is the advanced generative AI, which will be introduced in the first experiment.

5.2. RQ1. Effectiveness in Building Multi-step Attacks

We first evaluate whether the Attack Action Linking Model can effectively model the relationships between attack actions and automatically generate attack chains. Since no existing work can link third-party attack tools into executable, multi-step cyberattacks, we decided to compare AURORA with the currently most powerful generative AIs. We used AURORA, GPT-4o, GPT-o1, Deepseek-v3, and llama-3.3 to generate attack chains, with the results shown in Figure 6. The difference is that when using AURORA, we use the predicates from the Attack Action Linking Model to model preconditions and effects of attack actions. In contrast, we ask those LLMs to analyze the preconditions and effects of the same set of attack actions and convert them into PDDL predicates based on their knowledge and reasoning capability without any restrictions. For a fair comparison, we integrated basic predicates into the baseline methods, including operating system requirements, CVE information, and Metasploit exploit-payload module mappings, since existing attack planning works [39], [42], [50], [51], [52], [53], [54], [55] also consider these relationships. We also applied our design in Figure 5 to the LLMs to keep the consistency across the generated predicates.

The evaluation encompasses five metrics: 1) the number of predicates generated by the models to describe the preconditions and effects; 2) the number of normally functioning attack chains, defined as chains where: a. all actions have no arguments and preconditions missing, b. all actions execute successfully in the built environments, and c. all actions are meaningfully connected; 3) the average length of generated attack chains - we prefer longer attack chains since we want to emulate the multi-step attacks like real-world APT attacks; 4) correctness rate, measured as the proportion of normally functioning attack chains among all generated chains; and 5) the total number of attack actions

involved in the normally functioning attack chains - we hope to see a large number since we want to integrate more diverse attack actions.

The results in Figure 6 show AURORA leads significantly in all metrics (except the number of generated predicates). Based on the same set of attack actions, AURORA generates 247 normally functioning attack chains - twice as many as the most advanced GPT-01 model. The attack chains generated by AURORA have an average length of 4.98 actions, while other LLMs only connect around two actions on average. AURORA achieves 79% accuracy, three times higher than the best-performing baseline. Additionally, AURORA incorporates a wider variety of attack actions in its generated chains.

The reason why AURORA generates longer attack chains with higher accuracy and diversity is the proposed Attack Action Linking Model, especially the executor predicates. The executor predicates can effectively link abundant attack actions across different attack tools. For instance, the attack action set only has 20 attack actions (commands) provided by a Sliver session. However, by modeling the effects of the `msf` and `cmd` command provided by a Sliver session using predicates such as “`msf_payload_executor`” and “`powershell_executor`”, AURORA can immediately expand the attack chain with thousands of attack actions from Metasploit payload modules and PowerShell scripts in Atomic Red Team. However, we find existing work rarely considers the executor in planning multi-step cyberattacks and LLMs also fail to define the concept of executor when trying to generate predicates.

We also analyzed the reasons for the failed attack chains. The first reason is the inaccurate analysis on preconditions and effects by LLM, especially for attack actions involving complex parameters. For example, some Metasploit payload modules require more than ten arguments, with some being optional. LLMs usually struggle to accurately identify all precondition and effect predicates in such cases, causing some actions, although included in an attack chain, to fail during execution due to missing parameters or dependencies. We evaluate this issue in §5.5.2. The second reason is coarse-grained, inaccurate attack tool documentation, especially for open-sourced libraries such as Atomic Red Team. This issue requires developers to provide more detailed and accurate documentation. The third reason is the challenge of perfectly reproducing vulnerable environments for certain attacks. Our attack environment builder cannot configure specific software versions - a limitation we identify as a direction for future work (§6.3).

5.3. RQ2. Diversity of the Cyberattack Dataset

In this section, we evaluate the scale and diversity of the cyberattack dataset built by AURORA. Our evaluation consists of two parts: first, we compare the number of generated attack chains (scale) against the baselines, and second, we assess the number of distinct attack actions, MITRE tactics, and MITRE techniques covered (diversity). Please note that although we list eight baselines in Table 2,

TABLE 2: Comparison in terms of cyberattack dataset scale and TTP diversity.

	Attack Chains	Attack Actions	MITRE Techniques	MITRE Tactics	Execution Commands
APT Simulation [75]	13	N/A ¹	N/A ¹	N/A ¹	N
MITRE Eval. [74]	16	619	152	12	Y
PurpleSharp [28]	25	181	30	8	Y
Atomic Red Team [24]	N/A ²	1529	309	13	Y
Viper [68]	N/A ²	97	-	10	Y
SVED [76]	N/A ²	4000+	262	12	Y
Lore [41]	-	-	68	10	Y
ChainReactor [30]	-	31	-	-	N
AURORA	1056	5555	327	14	Y

¹ The baseline only describes general attack simulation plans, making it difficult to extract independent attack actions and identify corresponding MITRE ATT&CK TTPs.

² Cannot construct multi-step cyberattacks.

“-” means not clearly reported.

none of them has the capability of automatically creating a multi-step cyberattack dataset. APT Attack Simulation [75], MITRE Evaluation [74], and PurpleSharp [28] rely on human experts to craft attack chains. Viper [68], Atomic Red Team [24] and SVED [76] only provide atomic, isolated attack capabilities for testing, without forming them into attack chains. Lore [41] and ChainReactor [30] are the SOTA studies in automated red teaming and penetration testing, which are used to generate a single attack chain with the highest success rate on a given target. As shown in Table 2, AURORA surpasses all baselines in terms of dataset scale and diversity. By defining and linking modularized attack actions, AURORA can obtain more attack chains through (meaningful) permutations of attack actions. AURORA also has the largest diversity in terms of actions, techniques, and tactics since it merges the attack tools from different sources. Systems like ChainReactor lack specific attack commands and tools in their actions, relying instead on general conceptual actions, which further reduces their action number.

5.4. RQ3. Mimicry Attackers in CTI Reports

In this section, we evaluate if AURORA can effectively mimic the real-world attackers in CTI reports. We design the reward function to prioritize the attack actions mentioned in the reports during planning. We first review the generated attack chains, evaluating changes in the number of attack actions mentioned in CTI reports before and after incorporating the reward function design. We then present a case study to demonstrate how the reward function helps mimic the behaviors. We extracted TTPs from 50 CTI reports and assigned rewards to attack actions based on them. For each CTI report, we generate two sets of attack chains: one with actions that have rewards assigned based on that report, and another using the same action set but without rewards in planning.

Fig 7 shows how incorporating the reward function affects CTI report alignment. We count the number of attack actions in the generated attack chains whose corresponding MITRE techniques also appear in the reports. The blue line

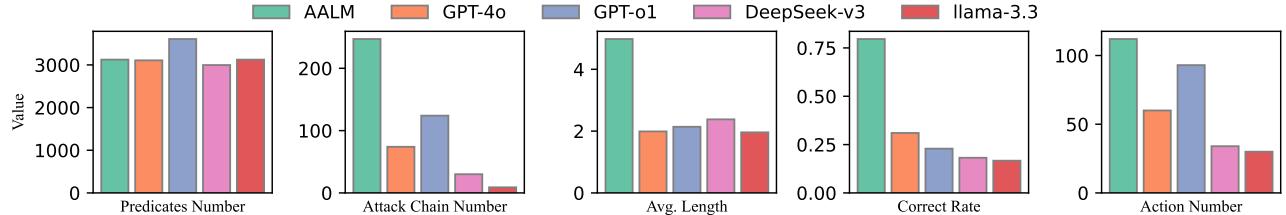


Figure 6: Comparison between the attack chains generated by the baselines and AURORA with Attack Action Linking Model.

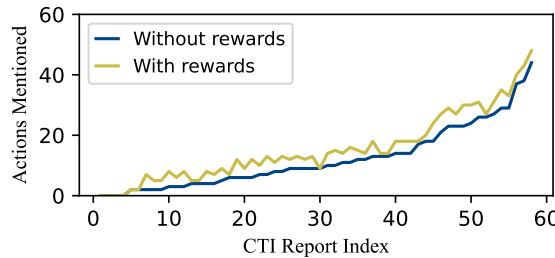


Figure 7: Comparison on the number of attack actions in generated attack chains that are also mentioned in CTI reports (with/without using reward function during planning).

represents results from the system without the reward function, while the golden line shows results from the system with the reward function mechanism. The figure demonstrates that the reward function effectively guides AURORA to prioritize attack actions documented in the reports during planning, resulting in a higher number of report-mentioned actions appearing in the generated attack chains. For some reports, we observed that the reward function had no impact. This happened either because the TTPs mentioned in these reports lacked corresponding attack actions, or because the attack planner coincidentally selected attack chains mentioned in the reports even without considering the rewards.

Figure 8 illustrates the impact of the reward function using two generated attack chains. Due to space constraints, other action details are omitted. Both chains achieve the same objective: obtaining a Meterpreter session on the target host and performing post-exploitation commands based on that. One way is exploiting the Apache Struts vulnerability (CVE-2016-3087) on the target host and executing a Meterpreter payload, which is chosen by the planner without the reward function. However, given the CTI report profiling a real attacker, Emotet Campaign, we observe that they typically gain initial access and execution through phishing email attachments and user interaction. As a result, the action *User download and execute the phishing attachment* receives a higher reward and is thus prioritized during planning. Thus, the planner opts for this longer attack chain, which aligns better with the profile with a higher total reward. This example shows how AURORA emulates the specific attackers in CTI reports with the help of the reward function.

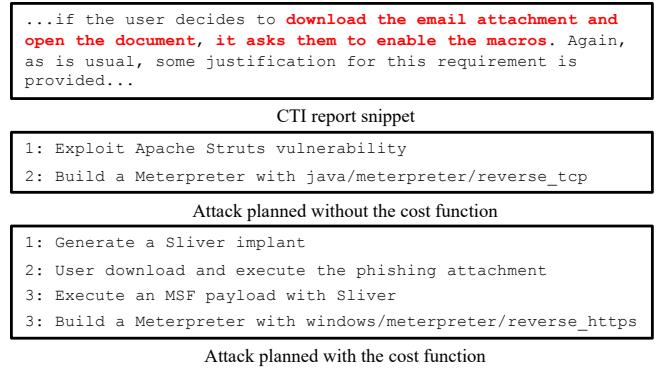


Figure 8: Comparison of two planned attacks with/without the cost function in the PDDL domain.

5.5. RQ4. Accuracy and Reliability of LLM

In AURORA, we use LLM to help us: 1. map attack actions from third-party attack tools to MITRE tactics and techniques; 2. automatically generate attack action preconditions and effects based on the Attack Action Linking Model; and 3. extract MITRE techniques from CTI reports, which - combined with the results from 1 - guide AURORA in emulating specific attack groups. In this section, we evaluate the accuracy and reliability of LLM and the impact of our improvement on vanilla LLMs.

5.5.1. Accuracy of Mapping Attack Actions to MITRE ATT&CK Tactics and Techniques. To enhance accuracy, we applied different techniques to the vanilla LLM model (GPT4o), including prompt engineering (PE) and fine-tuning (FT). We selected a set of 100 attack actions, independently annotated by two graduate students in security major, to serve as the ground truth. Using this dataset, we evaluated the accuracy in identifying MITRE tactics and techniques, as well as overall correctness. Here, correctness refers to the quality of the output, including correct spelling, completeness, and consistency between the technique name, technique ID, and corresponding tactic.

As illustrated in Figure 9, applying prompt engineering to the vanilla LLM significantly improved the accuracy of tactic and technique identification and, more notably, eliminated incorrect outputs. Surprisingly, we observed a decline in performance after fine-tuning the model - regardless of whether prompt engineering was used. The drop is due to the fact that the official MITRE ATT&CK dataset maps each

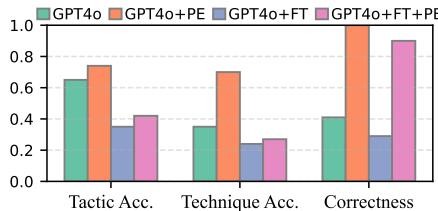


Figure 9: Comparison of performance in mapping attack actions to MITRE ATT&CK tactics and techniques.

attack step to a single technique, whereas in real attacks, a single action executed by an attacker often involves multiple techniques and tactics. As a result, fine-tuning with these one-to-one mappings hurts the identification accuracy. It highlights a key limitation of the MITRE ATT&CK TTP model in capturing the complexity of real-world, multi-faceted attack actions.

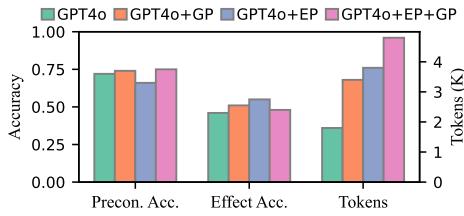


Figure 10: Comparison of performance in assigning preconditions and effects according to attack action linking model.

5.5.2. Accuracy of Analyzing Preconditions and Effects of Attack Actions. Figure 10 shows the accuracy in analyzing preconditions and effects of attack actions using LLMs. We labeled the preconditions and effects of 100 attack actions manually as the ground truth. The key question is how to teach LLM to accurately label the preconditions and effects according to the Attack Action Linking Model designed in §3. In order to do that, we use two different strategies in prompt engineering, including putting generating guidance in prompts (GP) and putting examples in prompts (EP). For GP, we wrote detailed descriptions of the Attack Action Linking Model in the prompt, hoping this would help LLM understand what kinds of preconditions and effects we care about when forming attack chains. For EP, we added three examples of attack actions and their corresponding preconditions and effects in the prompt. According to the results, adding detailed descriptions to the prompts (GP) led to a slight improvement in labeling preconditions and effects (though this came with increased costs: we needed more tokens). When adding example question-answer pairs to the prompts, results fluctuated, showing improved accuracy in effect identification but slightly decreased accuracy in precondition identification. Overall, however, we did not observe significant improvement in precondition and effect analysis from either strategy, indicating that LLM still needs

TABLE 3: 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%

improvement in analyzing preconditions and effects. We regard this as future work, which is also discussed in §6.1.

5.5.3. Accuracy of Report Analyzer. We also evaluated the performance of the report analyzer, which extracts the MITRE ATT&CK TTPs from the reports. We aimed to answer the following questions: 1. Can LLM accurately extract all TTPs from the reports? 2. Can LLM provide the correct description for each TTP it finds? To evaluate these two questions, we used CTI reports from CISA [69] as input, which provides the ground truth for verification. We evaluated the precision and recall of the TTP extraction. We also verified the correctness, which is defined as how many TTPs have correct descriptions. We compared the extraction accuracy of AURORA with one of the recent attack pattern extractors LADDER [34], which utilizes traditional NLP techniques.

As shown in Table 3, the report analyzer of AURORA achieves higher precision than the baseline, nearing or exceeding 80% for almost all reports. And the average recall is 69%. We scrutinize the descriptions of each extracted technique to evaluate the correctness. The result also shows that AURORA provides a more correct analysis 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 or ambiguous descriptions, such as generating an incomplete sentence: “exe control mechanism” as the description of System Binary Proxy Execution (T1218). More detailed analysis of the results can be found in Appendix E.1.

5.6. RQ5. Costs of AURORA

AURORA aims to reduce the time and domain expertise required for cyberattack emulation. In this section, we assessed the time and monetary costs of AURORA. The results in Table 4 show that mapping an attack action to its corresponding TTPs and analyzing its preconditions and effects takes less than 10 seconds on average (1.6s + 8.2s). This

means AURORA can analyze all attack actions within a day (this analysis only needs to be performed once, as the results can be reused). The time required to analyze one CTI report is approximately ten minutes (26.1s for TTP extraction and 640.9s for reward function embedding). Generating a single attack chain, setting up the environments, and executing the attack chain semi-automatically takes less than three minutes in total. Specifically, with the Python APIs of third-party attack tools, 99.8% of the attack actions in this paper can be executed using Python scripts. We also evaluate the cost of using LLM. Based on the price of the model we used (GPT-4o and text-embedding-ada-002), all attack chains in this paper cost less than 210 USD in total. The result shows that AURORA is economical in terms of both time and money.

TABLE 4: Average time and monetary costs of AURORA.

	Time	LLM Costs	
		Tokens	Cost(USD)
Attack Tool Analyzing - AALM ¹	1.6s	4.3K	0.011
Attack Tool Analyzing - MITRE Info ¹	8.2s	6.52K	0.018
CTI Report Analyzing ²	26.1s	16.8K	0.055
Reward Function using Embedding ²	640.9s	95.945K	0.010
Attack Planning ³	80.5s	0	0
Environment Setup ³	20.0s	0	0
Attack Execution ³	80.0s	0	0

¹ per attack action. ² per report. ³ per attack chain.

5.7. RQ6. Can cyberattacks generated by AURORA help evaluate detection systems?

In this section, we show the usage of AURORA by executing the constructed cyberattacks against three advanced APT detectors, NODOZE [17], PROVDETECTOR [77], and FLASH [5]. We selected these systems because they provide open-source implementations. A more comprehensive benchmarking study lies outside the scope of this paper. We encourage more security vendors to utilize AURORA to test the defense capabilities of their systems. We selected 18 attack chains, deployed virtual machines, and executed them. The execution of each attack chain took approximately 2 minutes, during which we collected system traces via ETW and converted them into provenance graphs. We simulated two hours of normal user behavior as training data. We labeled the system entities involved in the attacks according to the attack plan and evaluated the accuracy of these systems in detecting attack-related entities.

TABLE 5: Comparison of detection performance against attacks generated by AURORA.

	NODOZE	PROVDETECTOR	FLASH
True Positive Rate	49/63	6/63	20/63
False Positive Rate	3.39%	0.67%	0.16%

Table 5 shows the comparison of detection rates: NODOZE outperforms two other methods in true positive rate while having more false alarms in these detection

scenarios, while PROVDETECTOR requires further tuning to achieve better detection accuracy. This evaluation demonstrates the key utility of AURORA: generating cyberattacks to benchmark defense systems and identify weaknesses.

6. Limitations and Future Work

6.1. Improve the Accuracy of LLMs

We used LLMs to analyze the preconditions and effects of attack actions according to the Attack Action Linking Model. However, the experiment results show that even the best-performing LLM struggled to generate consistent, cohesive, and accurate predicates for preconditions and effects. A key future research question is how to enable LLMs to truly understand the Attack Action Linking Model and accurately label preconditions and effects. We believe this is an unavoidable question in “fully automated” attack emulation and requires deeper integration between LLMs and cybersecurity knowledge.

6.2. Completely Automated Attack Execution

AURORA can semi-automatically execute the generated attack chains thanks to the Python API of some third-party attack tools. However, certain constraints, such as user interaction and complex argument setting, still impede fully automated attack execution. For example, some actions need user interactions in the web browser. Some exploit modules need us to set the arguments manually (because LLM cannot understand each argument). LLM agents for tool use might be the answer to this problem.

6.3. Automated Environment Building

As mentioned in §4.4, building attack emulation environments is still an open question. AURORA can only utilize existing virtual machines, which are non-customizable. For example, we cannot customize virtual machines with specific combinations of software and vulnerabilities needed to execute an attack chain, which limits the choices of attack chains. Infrastructure as Code (IaC) tools like Terraform, Vagrant, and Ansible allow users to write descriptive IaC codes instead of dealing with various hardware and software. A possible research direction is leveraging the capability of generative AI to edit IaC configuration files [78]. But here are some challenges: (1) How to know and describe the emulation environment exactly? (2) How to guarantee the accuracy of IaC code generation?

7. Ethical Issue

We take the ethical issues in attack emulation seriously. First, the Attack Tool Analyzer and Attack Report Analyzer can only analyze existing attack tools and reports without involving any zero-day attack tool, technique, or weapon development. Our goal is to leverage existing public

knowledge of cyberattacks to enhance defensive capabilities. Secondly, we commit to releasing AURORA publicly upon the acceptance of this paper. We hope to collaborate with more security professionals to refine the Attack Action and Attack Action Linking Model, aiming to establish an open-source standard that empowers defenders to better simulate adversary behavior and confine the attackers.

8. Conclusion

In this paper, we first propose an attack action linking model for multi-step cyberattack automation. We then introduce AURORA, an automated cyberattack emulation system that addresses the challenges of integrating attack tools, linking attack steps, and mimicking CTI reports leveraging classical planning and LLMs. Using existing attack tools and reports, AURORA generates multi-step attack chains, sets up emulation environments, and semi-automatically executes these attack chains in the environments. Our evaluation results demonstrate that AURORA constructs higher-quality attack chains than the baselines within minutes. We have constructed and published a cyberattack dataset containing 1,000 attack chains.

References

- [1] S. Choi, J.-H. Yun, and B.-G. Min, “Probabilistic attack sequence generation and execution based on mitre att&ck for ics datasets,” in *Cyber Security Experimentation and Test Workshop*, 2021, pp. 41–48.
- [2] K. Satvat, R. Gjomemo, and V. Venkatakrishnan, “Extractor: Extracting attack behavior from threat reports,” in *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021, pp. 598–615.
- [3] Z. Li, J. Zeng, Y. Chen, and Z. Liang, “Attackg: Constructing technique knowledge graph from cyber threat intelligence reports,” in *European Symposium on Research in Computer Security*. Springer, 2022, pp. 589–609.
- [4] Y. Takahashi, S. Shima, R. Tanabe, and K. Yoshioka, “[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.
- [5] M. U. Rehman, H. Ahmadi, and W. U. Hassan, “Flash: A comprehensive approach to intrusion detection via provenance graph representation learning,” in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 139–139.
- [6] S. Li, F. Dong, X. Xiao, H. Wang, F. Shao, J. Chen, Y. Guo, X. Chen, and D. Li, “Nodlink: An online system for fine-grained apt attack detection and investigation,” *arXiv preprint arXiv:2311.02331*, 2023.
- [7] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, “Kairos: Practical intrusion detection and investigation using whole-system provenance,” in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3533–3551.
- [8] Z. Jia, Y. Xiong, Y. Nan, Y. Zhang, J. Zhao, and M. Wen, “[MAGIC]: Detecting advanced persistent threats via masked graph representation learning,” in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 5197–5214.
- [9] A. Goyal, G. Wang, and A. Bates, “R-caid: Embedding root cause analysis within provenance-based intrusion detection,” in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3515–3532.
- [10] L. Wang, X. Shen, W. Li, Z. Li, R. Sekar, H. Liu, and Y. Chen, “Incorporating gradients to rules: Towards lightweight, adaptive provenance-based intrusion detection,” *arXiv preprint arXiv:2404.14720*, 2024.
- [11] F. Yang, J. Xu, C. Xiong, Z. Li, and K. Zhang, “[PROGRAPER]: An anomaly detection system based on provenance graph embedding,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4355–4372.
- [12] J. Zengy, X. Wang, J. Liu, Y. Chen, Z. Liang, T.-S. Chua, and Z. L. Chua, “Shadewatcher: Recommendation-guided cyber threat analysis using system audit records,” in *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 489–506.
- [13] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, “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*, 2019, pp. 1795–1812.
- [14] M. N. Hossain, S. Sheikhi, and R. Sekar, “Combating dependence explosion in forensic analysis using alternative tag propagation semantics,” in *IEEE Symposium on Security and Privacy (SP)*, 2020.
- [15] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, “Unicorn: Runtime provenance-based detector for advanced persistent threats,” *arXiv preprint arXiv:2001.01525*, 2020.
- [16] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, “Holmes: Real-time apt detection through correlation of suspicious information flows,” in *IEEE Symposium on Security and Privacy (SP)*, 2019.
- [17] W. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, “Nodoze: Combatting threat alert fatigue with automated provenance triage,” 01 2019.
- [18] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, “You are what you do: Hunting stealthy malware via data provenance analysis.” in *NDSS*, 2020.
- [19] B. Jiang, T. Bilot, N. El Madhoun, K. Al Agha, A. Zouaoui, S. Iqbal, X. Han, and T. Pasquier, “ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems,” in *Security Symposium (USENIX Sec’25)*. USENIX, 2025.
- [20] “Darpa engagement data,” <https://drive.google.com/drive/folders/1okt4AYElyBohW4XiOBqmsjwXsnUjLVf>.
- [21] “Streamspot data,” <https://github.com/sbustreamspot/sbustreamspot-data>.
- [22] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, “[ATLAS]: A sequence-based learning approach for attack investigation,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3005–3022.
- [23] Online, “Metasploit: The world’s most used penetration testing framework,” <https://www.metasploit.com/>, 2025.
- [24] ———, “Explore atomic red team,” <https://atomicredteam.io/>, 2025.
- [25] J. Xu, J. W. Stokes, G. McDonald, X. Bai, D. Marshall, S. Wang, A. Swaminathan, and Z. Li, “Autoattacker: A large language model guided system to implement automatic cyber-attacks,” *arXiv preprint arXiv:2403.01038*, 2024.
- [26] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, “Kairos:: Practical intrusion detection and investigation using whole-system provenance,” *arXiv preprint arXiv:2308.05034*, 2023.
- [27] “Mitre caldera,” <https://github.com/mitre/caldera>.
- [28] “Purple sharp,” <https://detectionlab.network/usage/purplesharp/>.
- [29] S. T. R. Team, “Splunk attack range,” 2024, accessed: 2024-07-01. [Online]. Available: https://github.com/splunk/attack_range
- [30] G. D. Pasquale, I. Grishchenko, R. Iesari, G. Pizarro, L. Cavallaro, C. Kruegel, and G. Vigna, “ChainReactor: Automated privilege escalation chain discovery via AI planning,” in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 5913–5929. [Online]. Available: <https://www.usenix.org/conference/usenixsecurity24/presentation/de-pasquale>

- [31] A. Applebaum, D. Miller, B. Strom, C. Korban, and R. Wolf, "Intelligent, automated red team emulation," in *Proceedings of the 32nd annual conference on computer security applications*, 2016, pp. 363–373.
- [32] J. Hoffmann, "Simulated penetration testing: From" dijkstra" to" turing test++"," in *Proceedings of the international conference on automated planning and scheduling*, vol. 25, 2015, pp. 364–372.
- [33] G. Husari, E. Al-Shaer, M. Ahmed, B. Chu, and X. Niu, "Ttpdrill: Automatic and accurate extraction of threat actions from unstructured text of cti sources," in *Proceedings of the 33rd annual computer security applications conference*, 2017, pp. 103–115.
- [34] M. T. Alam, D. Bhusal, Y. Park, and N. Rastogi, "Looking beyond iocs: Automatically extracting attack patterns from external cti," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, 2023, pp. 92–108.
- [35] M. R. Rahman, R. M. Hezaveh, and L. Williams, "What are the attackers doing now? automating cyberthreat intelligence extraction from text on pace with the changing threat landscape: A survey," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–36, 2023.
- [36] U. Kumarasinghe, A. Lekssays, H. T. Sencar, S. Boughorbel, C. Elvitigala, and P. Nakov, "Semantic ranking for automated adversarial technique annotation in security text," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 49–62.
- [37] X. Qiu, S. Wang, Q. Jia, C. Xia, and Q. Xia, "An automated method of penetration testing," in *2014 IEEE Computers, Communications and IT Applications Conference*. IEEE, 2014, pp. 211–216.
- [38] J. Zhao, W. Shang, M. Wan, and P. Zeng, "Penetration testing automation assessment method based on rule tree," in *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*. IEEE, 2015, pp. 1829–1833.
- [39] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 2–10.
- [40] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "Pentestgpt: An llm-empowered automatic penetration testing tool," *arXiv preprint arXiv:2308.06782*, 2023.
- [41] H. Holm, "Lore a red team emulation tool," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1596–1608, 2022.
- [42] C. Phillips and L. P. Swiler, "A graph-based system for network-vulnerability analysis," in *Proceedings of the 1998 workshop on New security paradigms*, 1998, pp. 71–79.
- [43] B. Kordy, L. Piètre-Cambacédès, and P. Schweitzer, "Dag-based attack and defense modeling: Don't miss the forest for the attack trees," *Computer science review*, vol. 13, pp. 1–38, 2014.
- [44] M. S. Barik, A. Sengupta, and C. Mazumdar, "Attack graph generation and analysis techniques," *Defence science journal*, vol. 66, no. 6, p. 559, 2016.
- [45] A. Happe and J. Cito, "Getting pwn'd by ai: Penetration testing with large language models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 2082–2086.
- [46] A. Happe, A. Kaplan, and J. Cito, "Evaluating llms for privilege-escalation scenarios," *arXiv preprint arXiv:2310.11409*, 2023.
- [47] G. Chen, L. Yang, R. Jia, Z. Hu, Y. Chen, W. Zhang, W. Wang, and J. Pan, "Language-augmented symbolic planner for open-world task planning," *arXiv preprint arXiv:2407.09792*, 2024.
- [48] Y. Ding, X. Zhang, S. Amiri, N. Cao, H. Yang, A. Kaminski, C. Esselink, and S. Zhang, "Integrating action knowledge and llms for task planning and situation handling in open worlds," *Autonomous Robots*, vol. 47, no. 8, pp. 981–997, 2023.
- [49] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson *et al.*, "Pddl— the planning domain definition language," *Technical Report, Tech. Rep.*, 1998.
- [50] B. Schneier, "Modeling security threats," *Dr. Dobb's journal*, vol. 24, no. 12, 1999.
- [51] M. S. Boddy, J. Gohde, T. Haigh, and S. A. Harp, "Course of action generation for cyber security using classical planning," in *ICAPS*, 2005, pp. 12–21.
- [52] Q. Li, R. Wang, D. Li, F. Shi, M. Zhang, and A. Chattopadhyay, "Dynpen: Automated penetration testing in dynamic network scenarios using deep reinforcement learning," *IEEE Transactions on Information Forensics and Security*, 2024.
- [53] K. Tran, A. Akella, M. Standen, J. Kim, D. Bowman, T. Richer, and C.-T. Lin, "Deep hierarchical reinforcement agents for automated penetration testing," *arXiv preprint arXiv:2109.06449*, 2021.
- [54] J. Chen, S. Hu, H. Zheng, C. Xing, and G. Zhang, "Gail-pt: An intelligent penetration testing framework with generative adversarial imitation learning," *Computers & Security*, vol. 126, p. 103055, 2023.
- [55] M. C. Ghanem, T. M. Chen, and E. G. Nepomuceno, "Hierarchical reinforcement learning for efficient and effective automated penetration testing of large networks," *Journal of Intelligent Information Systems*, vol. 60, no. 2, pp. 281–303, 2023.
- [56] "Att&ck matrix for enterprise," <https://attack.mitre.org/>.
- [57] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," *arXiv preprint arXiv:1306.4044*, 2013.
- [58] D. Miller, R. Alford, A. Applebaum, H. Foster, C. Little, and B. Strom, "Automated adversary emulation: A case for planning and acting with unknowns," *MITRE CORP MCLEAN VA MCLEAN, Tech. Rep.*, 2018.
- [59] BishopFox, "Sliver," 2024, accessed: 2025-04-01. [Online]. Available: <https://github.com/BishopFox/sliver>
- [60] Fortra, "Software for adversary simulations and red team operations," 2024, accessed: 2025-04-01. [Online]. Available: <https://www.cobaltstrike.com/>
- [61] Rapid7, "Metasploit documentation of meterpreter," 2024, accessed: 2025-04-01. [Online]. Available: <https://docs.metasploit.com/docs/using-metasploit/advanced/meterpreter/meterpreter.html>
- [62] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.
- [63] "Ludus: The easiest way to deploy dev/test infrastructure," <https://ludus.cloud/>.
- [64] B. Green, R. Derbyshire, W. Knowles, J. Boorman, P. Ciholas, D. Prince, and D. Hutchison, "{ICS} tested tetris: Practical building blocks towards a cyber security resource," in *13th USENIX workshop on cyber security experimentation and test (CSET 20)*, 2020.
- [65] B. Ruan, J. Liu, C. Zhang, and Z. Liang, "Kernjc: Automated vulnerable environment generation for linux kernel vulnerabilities," in *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, ser. RAID '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 384–402. [Online]. Available: <https://doi.org/10.1145/3678890.3678891>
- [66] "Atomic red team," <https://github.com/redcanaryco/atomic-red-team>. [Online]. Available: <https://github.com/redcanaryco/atomic-red-team>
- [67] "Living off the land binaries, scripts and libraries <https://lolbas-project.github.io/>." [Online]. Available: <https://lolbas-project.github.io/>
- [68] FunnyWolf, "Viper," 2024, accessed: 2024-11-01. [Online]. Available: <https://github.com/FunnyWolf/Viper?tab=readme-ov-file>
- [69] CISA, "Cybersecurity alerts advisories," 2024, accessed: 2024-07-01. [Online]. Available: <https://www.cisa.gov/news-events>

- [70] Symantec, “Symantec enterprise blogs threat intelligence,” 2024, accessed: 2024-07-01. [Online]. Available: <https://symantec-enterprise-blogs.security.com/threat-intelligence/clasiopa-materials-research>
- [71] T. D. REPORT, “The dfir report,” 2024, accessed: 2024-07-01. [Online]. Available: <https://thedfirreport.com/>
- [72] AI-Planning, “pddl,” 2024, accessed: 2024-11-01. [Online]. Available: <https://github.com/AI-Planning/pddl>
- [73] A. I. G. U. of Basel, “Fast downwards,” 2024, accessed: 2024-11-01. [Online]. Available: <https://github.com/aibasel/downward>
- [74] M. Engenuity, “Cybersecurity: Att&ck® evaluations,” 2024, accessed: 2024-07-01. [Online]. Available: <https://mitre-engenuity.org/cybersecurity/attack-evaluations/>
- [75] S3N4T0R-0X0, “Apt attack simulation,” 2024, accessed: 2025-03-01. [Online]. Available: <https://github.com/S3N4T0R-0X0/APT-Attack-Simulation>
- [76] H. Holm and T. Sommestad, “Sved: Scanning, vulnerabilities, exploits and detection,” in *MILCOM 2016-2016 IEEE Military Communications Conference*. IEEE, 2016, pp. 976–981.
- [77] Q. Wang, W. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter, and H. Chen, “You are what you do: Hunting stealthy malware via data provenance analysis,” 01 2020.
- [78] T. Chanus and M. Aubertin, “Llm and infrastructure as a code use case,” *arXiv preprint arXiv:2309.01456*, 2023.

Appendix A. PDDL Problem File Example

Figure 11 shows an example of the PDDL problem file. The problem file is organized from top to bottom with the following components: the domain name, PDDL syntax requirements, objects needed for the plan (for brevity, only objects used in the definition are included, omitting those needed in intermediate steps), the initial state, the goal state, and the optimization objective.

```
(define (problem data-deletion-attack)
(:domain attack-planner-auto)
(:requirements :action-costs :equality :strips :typing)
;; Omit other objects used in planning for succinctness
(:objects data_path - path data_file - file target - host)
(:init
  ;; Set Total Reward to 0 before planning starts
  (= (total-reward) 0)
  (OS_windows target)
  (file_exists data_path data_file target))
(:goal (file_deletion data_file))
(:metric maximize (total-reward))
)
```

Figure 11: An illustrative example of the PDDL problem file.

Appendix B. Attack Action Linking Model

Figure 12 shows an example of an attack action defined in §3.1.

```
Name: Keylogger Activation
UUID: 22de2d50-3abb-4e31-a131-206d88240565
Source: Meterpreter
ID: T1056.001
Supported Platforms:
  - windows
  - linux
  - macos
Tactics:
  - Collection
  - Credential Access
Technique: Input Capture: Keylogging
Description: The command `keyscan_start <options>` is used within a Meterpreter session to initiate a keylogger, which records keystrokes on the target machine.
Execution:
  Executor: Mettrepreter session
  command: keyscan_start <options>
Preconditions:
  - (or
    (OS_windows ?t - host)
    (OS_linux ?t - host)
    (OS_macos ?t - host)
  )
  - (meterpreter_session ?e - executor ?t - host)
  - (elevated ?e - executor)
Effects:
  - (keylogger_data_saved ?f - file ?t - host)
  - (file_exists ?f - file ?p - path ?t - host)
```

Figure 12: An example of attack action

Due to space constraints in §3.2, we do not provide exhaustive details of the Attack Action Linking Model and its predicate examples. Here we present the formats and examples of predicates used for linking attack actions and generating attack chains using classical planning.

```
# Environment
## Operating Systems: Indicates the operating system running on the target machine.

Examples:
(OS_windows ?target - host)
(OS_linux ?target - host)
(OS_macos ?target - host)

## Vulnerabilities: Indicates the vulnerabilities existing on the target machine.

Format:
(CVE_{cve_id}_exists ?target - host)

Examples:
(CVE-2004-2687_exists ?target - host)

## Software: Indicates the software running on the target machine.

Format:
(software_{software_name}_exists ?target - host)

Examples:
(MS_word_exists ?target - host)

# Executor
## Executor Type: Indicates the type of the executor.

Examples:
```

```

(command_prompt_executor ?eID -> executor ?
    target = host)
(powershell_executor ?eID -> executor ?target =
    host)
(bash_executor ?eID -> executor ?target = host)
(sliver_session ?eID -> executor ?target = host
    )
(meterpreter_session ?eID -> executor ?target = host)

## Privilege Level
    (elevated_executor ?exeID -> executor)

# Payload
## Payload Usage: Indicates the type of the payload.
Examples:
    (sliver_implant_payload ?p -> payload ?target = host)

## Payload Type: Indicates the type of the payload .
Examples:
    (file_payload ?p -> payload ?f -> file)
    (shellcode_payload ?p -> payload ?s -> shellcode
        )
    (command_payload ?p -> payload ?c -> command)

## Payload Handler Type: Indicates the type of the payload handler.
Format:
    ({payload_name}_payload_handler ?p -> payload)

Examples:
    (MSF-windows-
        meterpreter_reverse_http_payload_handler ?p -> payload)

## Payload Operation: Indicates the operation regarding the payload.
Examples:
    (payload_executed ?payload -> payload ?target = host)
    (payload_handler_set ?payload -> payload)
    (payload_executed_as_root ?payload -> payload ?target = host)

# Process
## Process Status
Examples:
    (process_running ?p -> process ?t -> host)
    (process_terminated ?p -> process)

# File and Directory
## File Type
Examples:
    (exe_file ?f -> file)
    (dll_file ?f -> file)
    (doc_file ?f -> file)

## File/Directory Operation: Indicates the operation regarding the file and directory.
Examples:
    (file_exists ?path -> path ?file -> file ?target = host)
    (dir_exists ?path -> path ?dir -> dir ?target = host)
    (file_executed ?file -> file ?target = host)
    (file_executed_as_root ?file -> file ?target = host)
    (file_deleted ?file -> file ?target = host)

```

```

(file_permission_modified ?file -> file ?target = host)

# User
## User Status
Examples:
    (user_exists ?user -> user ?target = host)

## User Type
Examples:
    (root_user ?user -> user)

# Credential
## Credential Known: Indicates some credentials on the target machine is known by the attacker .
Format:
    ({service/software/account}_password_known ?a account ?p -> password ?target = host)

Examples:
    (email_password_known ?a account ?p -> password ?target = host)
    (ssh_password_known ?a account ?p -> password ?target = host)

# Information
## Information Known: Indicates some information of the target machine is known by the attacker .
Format:
    ({info_details}_info_printed ?target = host)

Examples:
    (ip_info_known ?ip -> ip ?target = host)
    (vul_port_known ?port -> port ?target = host)

# Data
Format:
    ({data_source}_data_printed ?target = host)
    ({data_source}_data_saved ?file -> file ?target = host)
    ({data_source}_data_deleted ?target = host)

Examples:
    (screenshot_data_saved ?file -> file ?target = host)

# Others
Examples:
    (system_reboot ?target = host)

```

Listing 1: Predicate set of the attack action linking model used in this paper.

Figure 13 shows a simple example of an attack chain. In the main paper, we demonstrated how this chain's actions are connected through AALM predicates. Here, we explain each predicate in detail.

Appendix C. Prompts

In this section, we briefly summarize the strategy of prompt engineering during our usage of LLM in AURORA. The first strategy is focusing on technical details in the prompts. Ethical restrictions are a major concern for LLM

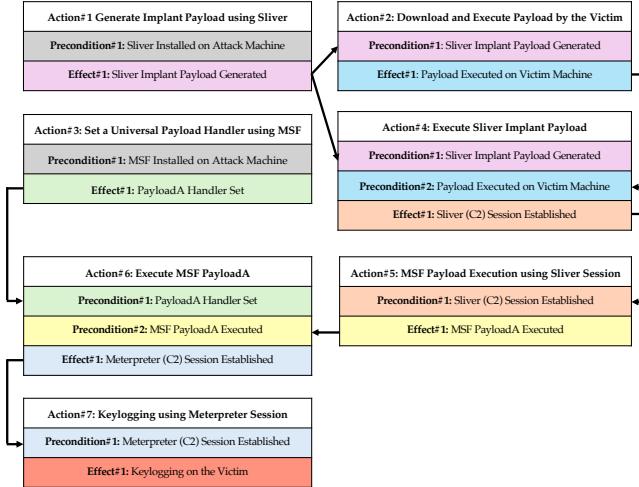


Figure 13: An example of the attack chain.

applications in cybersecurity; however, we have not encountered ethical restriction issues from LLM providers in our project. We attribute this to the structure of our prompts, which focus on technical specifics rather than the broader intent of the attack. Instead of revealing our intentions about the attack, we ask the LLM questions about specific technical aspects, such as the effect of a script or the pre-conditions of a command. Second, we use some examples in the prompts to guide LLM. Describing the requirements in a prompt can sometimes be challenging. We find it helpful to use a real example to illustrate what we want the LLM to accomplish. For instance, we use the predicates defined by human experts in [30] as an example when generating our own predicates.

CTI Report Analyzer Prompt:
You are a world class security engineer.
 Please analyze this CTI report, break down the attack into small steps, and map each step to the Tactic, Technique, and Procedure (TTP) of MITRE ATT&CK Matrix...
@/** The complete prompt will be published once accepted **@
 The list of TTPs from MITRE ATT&CK Matrix is here:
{technique_list}.
 Please output the results in the format of json. Please only output the texts in the json file. Do not add any preambles.\n
 Here is the output template: {output_template}.
 Here is the report: {report_info}.

Figure 14: Attack report analyzer prompt.

Appendix D. Details in the Design of AURORA

D.1. Searching for Attack Chains using PDDL Planners

In § 4.3, we introduced how we use PDDL-based classical planning algorithms to generate attack chains. It should be noted that most algorithms stop after finding the first plan. To find as many feasible attack paths as possible, we designed an attack path search algorithm that leverages the PDDL cost mechanism. 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.

Appendix E. Supplementary Experimental Results

E.1. Failure Cases in Report Analyzing

We notice some failure cases, e.g. reports 3, 7, and 9. There are some reasons: First, the ground truth sometimes does not completely list all techniques mentioned in the report. Upon deeper examination of reports with low precision, we find that some techniques identified by the LLM indeed exist in the report but are not mentioned in the ground truth (report 3 and 9), which implies LLM can sometimes outperform humans in identifying techniques from reports. Second, LLM makes mistakes when some natural language descriptions are ambiguous. For instance, in the case of ALPHV Blackcat ransomware, the report mentions that “affiliates communicate with victims via TOR, Tox, email, or encrypted applications” (to ask for the ransom). However, the LLM incorrectly identified this sentence as evidence of a command and control technique during the attack. Moreover, we observe that LLM tends to overly depend on certain keywords when interpreting entire sentences. Consequently, it might misidentify techniques associated with those keywords. For instance, in the case of Scattered Spider, 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).