NORTHWESTERN
UNIVERSITY

Computer Science Department

Technical Report
Number: NU-CS-2025-32

September, 2025

Beyond Evaluation: Towards Automated and Explainable Red Teaming

Xiangmin Shen

Abstract

Understanding how security defenses perform under real-world adversarial conditions remains a core challenge in enterprise security. While detection and response systems are widely deployed, organizations often lack structured, meaningful methods to assess their effectiveness. This dissertation begins by analyzing the MITRE ATT&CK Evaluations, a prominent adversary emulation framework, and introduces *Decoding MITRE ATT&CK Evaluations*, a graph-based system that reconstructs attack chains, models detection responses, and quantifies metrics such as protection latency, detection coverage, and alert connectivity. This analysis reveals a broader limitation: evaluation is constrained by the scarcity and rigidity of curated attack scenarios. To address this, the dissertation presents *PentestAgent*, a multi-agent penetration testing framework leveraging large language models and retrieval-augmented generation to automate reconnaissance, planning, and exploitation, and *AEAS* (Actionable Exploit Assessment System), which evaluates public exploit artifacts using structured feature extraction and language model reasoning to produce actionability scores and justifications. Together, these systems make red teaming more explainable, scalable, and operationally relevant by bridging evaluation, simulation, and planning for proactive security testing.

Keywords

System Security, AI for Security, Intrusion Detection, Penetration Testing

NORTHWESTERN UNIVERSITY

Beyond Evaluation: Towards Automated and Explainable Red Teaming

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Computer Science

By

Xiangmin Shen

EVANSTON, ILLINOIS

September 2025

ABSTRACT

Understanding how security defenses perform under real-world adversarial conditions remains a core challenge in enterprise security. While detection and response systems are widely deployed, organizations often lack the ability to evaluate their effectiveness in a structured and meaningful way. This dissertation begins by examining the MITRE ATT&CK Evaluations, a leading framework for adversary emulation that tests commercial Endpoint Detection and Response (EDR) products using multi-stage attack simulations. Despite the value of these evaluations, their publicly reported results are coarse and difficult to interpret, offering limited insight into detection quality, alert timing, or systemic blind spots.

To address this, the first part of the dissertation introduces Decoding MITRE ATT&CK Evaluations, a graph-based framework for reconstructing attack chains and modeling detection responses. The system quantifies protection latency, detection coverage, and alert connectivity, enabling a more nuanced and actionable analysis of EDR performance. However, this analysis also reveals a deeper limitation: evaluation efforts are constrained by the scarcity and rigidity of curated attack scenarios.

To overcome this limitation, the remainder of the dissertation focuses on facilitating the generation of high-quality red team simulations through automation and decision support. It first presents PentestAgent, a multi-agent penetration testing framework powered by large language models and retrieval-augmented generation. PentestAgent decomposes the attack workflow into specialized agents that collaborate to perform reconnaissance, planning, and exploitation. This system reduces manual effort and improves the realism and scalability of attack emulation.

Next, the dissertation introduces AEAS, the Actionable Exploit Assessment System. AEAS analyzes public exploit artifacts using structured feature extraction and language model reasoning

to produce actionability scores, severity assessments, and human-readable justifications. It helps red teamers and automated systems make more informed exploit selection decisions during the planning phase.

Together, these three systems address key limitations in how red teaming is interpreted, executed, and prioritized. Rather than attempting to generate novel attacks from scratch, this dissertation focuses on building the tools and frameworks that make red teaming more explainable, scalable, and practically useful. By bridging evaluation, simulation, and planning, it contributes to a more integrated and operationally relevant approach to proactive security testing.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Yan Chen, for his invaluable guidance, support, and encouragement throughout my PhD journey. His insights and expertise have been instrumental in shaping my research and academic growth. Beyond academic matters, Professor Chen has been a source of inspiration and motivation in my personal life as well. Inspired by his example, I took up running and completed my first marathon during my PhD.

I would also like to thank my committee members, Professor Xinyu Xing, Professor Han Liu, and Professor Zhenkai Liang, for their constructive feedback and support. Professors Xing and Liang also served on my qualifying exam committee, and I greatly appreciate the thoughtful guidance they provided during that stage of my doctoral work.

Special thanks go to Lingzhi Wang, a close collaborator and friend throughout my PhD. His dedication, insights, and encouragement have been invaluable both academically and personally. The countless hours we spent working together and supporting each other through challenging times are among my most cherished memories of this journey.

I would like to thank my colleagues and collaborators at Northwestern University, Stony Brook University, Zhejiang University, and Ant Group, including but not limited to: Professor R. Sekar, Xutong Chen, Xuechao Du, Xing Li, Mohammad Kavousi, Kaiyu Hou, Runqing Yang, Guannan Zhao, Haozheng Luo, Yuxiao Tang, Jiashui Wang, Wencheng Zhao, and Dawei Sun. It has been a privilege to work with such talented and dedicated individuals, and I am deeply appreciative of their contributions to my research and professional development.

I am especially grateful to my parents for their unwavering love, patience, and encouragement. Due to the pandemic and other challenges, I have not been able to return home to see them during

my PhD. I miss them dearly and look forward to reuniting soon. Their support has been a constant source of strength throughout my academic journey.

To my girlfriend, Yingzhe Zhu, I am grateful for your love and steadfast support. Your patience and understanding sustained me through the most challenging phases of my PhD, and your perspectives as a sociologist have enlightened the way I approach my research.

I would also like to thank my friends for their companionship and encouragement. Whether over shared dinners or ski trips, their friendship has provided joy and balance amidst the demands of graduate study.

Finally, I am thankful to Northwestern University and the City of Evanston. Spending ten years here for both my undergraduate and graduate studies has been a privilege, and I deeply appreciate the safe and welcoming environment that has been my home throughout this academic journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION AND BACKGROUND

1.1 Motivation

A central question in enterprise security operations is: *how effective are our defenses against real-world adversaries?* Despite the wide deployment of detection and response systems such as endpoint protection platforms, intrusion detection systems, and behavioral analytics, organizations often lack concrete answers. Detection metrics, coverage claims, and alert dashboards provide fragmented indicators, but they do not capture whether or how well a defense performs against complex, multi-stage attacks in practice.

To address this gap, the security community has increasingly turned to adversary emulation. One of the most prominent efforts in this space is the MITRE ATT&CK Evaluations, which simulate real-world APT campaigns against commercial Endpoint Detection and Response (EDR) products. These evaluations are valuable because they offer repeatable, well-documented attack scenarios and expose how EDR systems respond under adversarial pressure. However, my experience working with this dataset revealed a critical limitation: the results are presented in a binary and overly simplified format, making them difficult to interpret. Detections are recorded as present or absent, with little contextual information about alert timing, correlation across stages, or response quality.

This led to the first contribution of the dissertation. To make sense of adversary emulation data in a structured and actionable way, I developed a graph-based interpretation framework. This system reconstructs the complete attack chain and introduces formal metrics such as protection

latency, detection coverage, and alert connectivity. By capturing not only whether a technique was detected but how and when it was observed in the broader context of the attack, this framework enables more informative and comparative analysis across EDR products. The system has been used to expose systemic blind spots in commercial detection systems and to help defenders understand how specific gaps emerge across the kill chain.

However, this work also revealed a broader problem. The utility of defense evaluation is fundamentally limited by the scope and quality of the attacks used. While the MITRE evaluations are valuable, they are curated, static, and updated only periodically. For defenders to continuously evaluate and improve their systems, they need access to a broader set of realistic attack scenarios. These scenarios must be relevant to their environments, reflect current threat trends, and be flexible enough to support testing across different detection configurations. Achieving this requires not only the creation of attack content, but also the tools and frameworks that support red teamers in crafting, adapting, and deploying attacks efficiently and effectively.

This motivated a shift in the focus of the dissertation toward facilitating the generation and execution of high-quality attack simulations. A major obstacle in this space is the high cost of manual red teaming. Crafting realistic attack campaigns requires expert knowledge and significant effort, as red teamers must plan multi-step operations, identify appropriate tools, and interact with environments in a context-aware manner. These tasks are often performed manually or with brittle automation scripts, making it difficult to scale or integrate red teaming into continuous evaluation pipelines.

To address this challenge, I developed PentestAgent, a multi-agent penetration testing framework designed to assist with realistic and adaptive red team operations. PentestAgent is powered by large language models and retrieval-augmented generation, and decomposes the red team workflow into specialized agents responsible for reconnaissance, vulnerability discovery, planning, and

exploitation. Each agent operates semi-autonomously, reasoning about its current objectives and retrieving relevant information to inform its actions. The framework is designed not to replace human red teamers, but to support them by automating routine tasks and enabling more consistent and scalable adversary emulation.

Even with scalable execution, the planning phase remains critical. One of the most important planning decisions is exploit assessment. Public exploit repositories contain a mix of mature, functional exploits and incomplete or outdated proofs of concept, many of which are poorly documented or require extensive modification. Existing scoring systems such as CVSS and EPSS provide severity or likelihood estimates, but they do not reflect an exploit's actual usability in practice. This creates inefficiencies for both human and automated red teamers, who must sift through unreliable options.

To close this gap, I developed AEAS (Actionable Exploit Assessment System). AEAS combines large language model reasoning with structured feature extraction to evaluate public exploit artifacts for availability, functionality, and setup complexity. It produces actionability scores, severity assessments, and natural-language justifications, enabling red teamers, human or automated, to select high-quality, operationally viable exploits efficiently. By transforming unstructured exploit data into structured, interpretable intelligence, AEAS streamlines decision-making during attack planning.

Taken together, these three projects, Decoding MITRE Evaluations, PentestAgent, and AEAS, represent a unified effort to enhance red teaming through analysis, automation, and prioritization. Rather than aiming to generate new attacks from scratch, this dissertation focuses on building the tools and frameworks that facilitate realistic, explainable, and scalable red team simulation, thereby helping defenders continuously assess and improve their security posture.

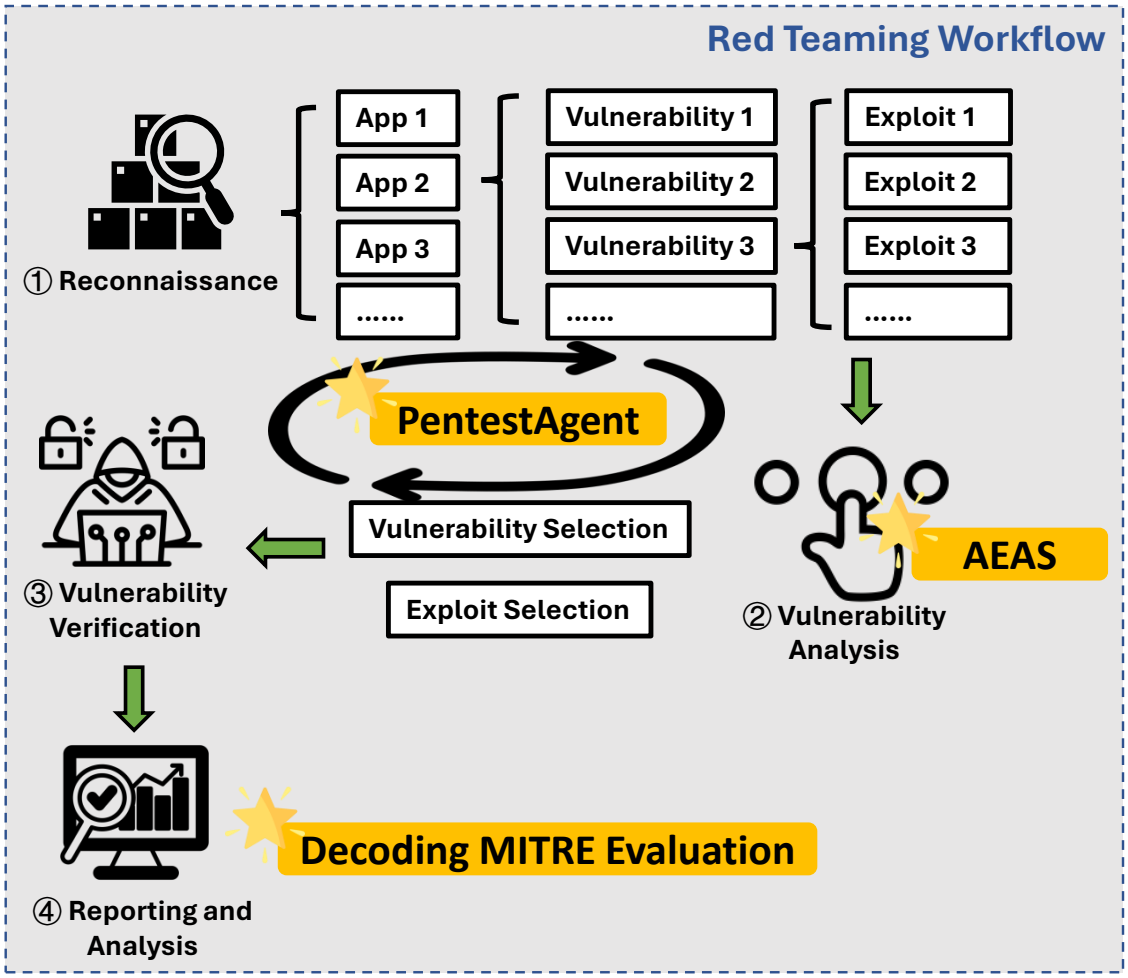


Figure 1.1: Red teaming workflow and contributions

1.2 Summary of Contributions

This dissertation introduces three systems that address key limitations in how red teaming is conducted, interpreted, and scaled in modern security practice. Each system targets a distinct challenge along the red teaming lifecycle, with a focus on improving the structure, efficiency, and transparency of offensive evaluation.

1. **Decoding MITRE ATT&CK Evaluations**, a graph-based framework for analyzing the results of adversary emulation exercises. This system is motivated by the limited interpretability of datasets like the MITRE ATT&CK Evaluations, where detection outcomes are recorded in coarse-grained formats that hinder meaningful analysis. Decoding MITRE reconstructs full attack chains from evaluation data and introduces structured metrics such as protection latency, detection coverage, and alert connectivity. The framework enables defenders and evaluators to reason more precisely about detection quality and systemic weaknesses, providing a foundation for rigorous comparison across commercial EDR products.

2. **PentestAgent**, a multi-agent penetration testing framework designed to facilitate scalable and realistic attack simulation. Traditional red teaming workflows involve significant manual effort and fragmented tooling, which restrict the frequency and reproducibility of engagements. PentestAgent addresses this by decomposing the attack lifecycle into specialized agents for reconnaissance, planning, and exploitation, each powered by large language models and retrieval-augmented generation. The system supports the automation of common attack tasks while preserving transparency and adaptability, allowing users to generate rich and customizable adversarial scenarios with minimal overhead.

3. **AEAS (Actionable Exploit Assessment System)**, a system that addresses the challenge

of exploit assessment during red team planning. Public exploit repositories contain artifacts of widely varying quality, creating inefficiencies in identifying reliable options. AEAS combines large language model reasoning with structured feature extraction to evaluate exploit availability, functionality, and setup complexity. It produces actionability scores, severity assessments, and natural-language justifications, supporting more informed and efficient decision-making for both human and automated red teamers.

Collectively, these systems advance the state of red teaming by enabling organizations to evaluate defenses more effectively, simulate attacks more realistically, and plan operations more intelligently. Rather than building attack content from scratch, this dissertation focuses on developing the infrastructure and tools that make red teaming more scalable, explainable, and operationally relevant.

1.3 Organization of the Dissertation

This dissertation is organized into five chapters. Following this introduction, Chapter 2 introduces Decoding MITRE ATT&CK Evaluations, a framework for interpreting adversary emulation results through graph-based modeling and structured metrics. This work serves as the initial motivation, revealing the limitations of existing evaluation methods. Chapter 3 presents PentestAgent, a multi-agent penetration testing framework that automates key phases of red teaming, including reconnaissance, planning, and exploitation, using language models and retrieval-based reasoning. Chapter 4 introduces AEAS, the Actionable Exploit Assessment System, which analyzes public exploit artifacts to produce actionability scores, severity assessments, and justifications, enabling more informed exploit selection during planning. Finally, Chapter 5 concludes the dissertation by synthesizing insights across all three systems and outlining future research directions in automated and explainable red teaming.

CHAPTER 2

DECODING THE MITRE ENGENUITY ATT&CK ENTERPRISE EVALUATION: AN ANALYSIS OF EDR PERFORMANCE IN REAL-WORLD ENVIRONMENTS

2.1 Introduction

The digital revolution dramatically changed human life and brings new risks into daily life. Driven by profit, attackers in cyberspace have organized increasingly sophisticated attacks that affect many organizations and large corporations such as Siemens, Target, and Equifax. These attacks resulted in millions of consumers' data being leaked [1]–[3] and other losses. Traditional network-based prevention and detection approaches can barely deal with these advanced attacks. Therefore, endpoint-based detection and response solutions (EDR) and extended solutions (XDR) receive extensive attention in academia and industry. The market capitalization for the top 37 EDR companies has reached over 320 billion U.S. dollars [4] by 2022, the market size of 3.7 billion U.S. dollars [5]. Meanwhile, both market capitalization and size are expected to grow fast.

As the number of homogeneous EDR products increases, it becomes increasingly difficult for users to choose the appropriate product. Fair third-party evaluations with detailed interpretations of results are therefore necessary. The challenges of conducting such evaluations are three-fold. Firstly, the evaluation methodology should be general enough to allow broad participation. The significance of the evaluation will be affected if its methodology only applies to a small set of security solutions. Secondly, the evaluation should perform realistic and various attack emulations. The attack emulations in the laboratory setting are simple and have little variety. Sometimes, the attack information is even known before the evaluation, making it possible for the defense team

to make ad-hoc configuration adjustments. Such attack emulations can not reveal the actual performance of endpoint security solutions against real-world threats. Finally, the evaluation results should be reported with comprehensive interpretation. Without appropriate metrics and objective interpretation, the evaluation results are hard to understand, possibly leading to biased interpretation.

Although much effort has been made toward establishing a norm for security solutions, there is still no substantial benchmark in the security field. Several for-profit companies and organizations present their own evaluation results [6], [7]. However, their methodologies are not transparent and could be biased for commercial reasons. In academia, several recent benchmark works [8]–[12] focus on generating new or improving existing datasets. Although they are crucial initial steps, equally-important interpretations of evaluation results are still missing. Several other benchmarking works [13], [14] attempt to expand the explainability of evaluation results. But their methodologies are specific to the target systems or platforms, making their evaluation methodologies and results not transferable.

To standardize security evaluations, the MITRE Corporation has been conducting annual APT emulations to evaluate various security solutions based on its ATT&CK framework [15] since 2018. In each round of evaluation, MITRE will select one or more real-world APT groups and reconstruct their typical attack chains in controlled environments. The EDR products will be deployed in these environments in advance. And the results will be collected and published by MITRE to summarize their performance. The results list the attack steps characterized by MITRE ATT&CK techniques in attack emulations. For each EDR system, MITRE publishes its detection and protection performance on attack steps. The detection performance is described with MITRE-defined detection categories, indicating the amount of contextual information the EDR system provides with the alarm. The protection performance is illustrated by the step at which the attack

is blocked.

While the datasets from MITRE's evaluation are valuable, the presentation of evaluation results has some apparent defects, preventing security practitioners from benefiting directly. These problems include missing whole-graph analysis, lacking comprehensive interpretation, and inconsistent evaluation framework. Concretely, MITRE only focuses on single-step detection results. However, the attacks that EDR systems fight against are sophisticated and involve multiple steps. EDR systems must consider the entire kill chain to provide satisfying detection and response services. Therefore, the whole-graph analysis capability is crucial in evaluating EDR systems. Apart from conducting attack emulations and collecting results, interpreting the results is equally or not more critical to EDR system evaluation. However, MITRE makes minimal effort in interpreting the results. Lacking comprehensive interpretation prevents users from getting direct insights from the results and could lead to biased interpretations for vendors and customers. MITRE has only conducted four evaluations so far. Understandably, its methodology has evolved, leading to several inconsistencies in the published results. Inconsistencies like these can be burdensome for users, forcing users to investigate the difference in every year's methodology.

To address these problems, we propose analysis methodologies on the MITRE evaluation dataset to perform fine-grained whole-graph analysis and holistic assessments of EDR systems' capabilities. Then, we apply our methodology to analyze MITRE evaluation datasets. We investigate all attack scenarios and construct causal relationship attack graphs to present causal relationships between attack steps. We evaluate EDR systems' attack reconstruction capability by conducting the connectivity analysis, examining whether the EDR system can reconstruct the complete attack kill chain. We also assess EDR systems' response capability via the effectiveness analysis. In the effectiveness analysis, we use protection performance as an indicator. Specifically, we examine at which step each EDR system responds to the attack and determine if the EDR system is effectively

protecting the host. Moreover, we discuss the evaluation results from several practical perspectives to measure the detection and protection performance of individual techniques and EDR systems, including detection coverage, detection confidence, detection quality, data source, and compatibility. We also investigate the trend of performance change from these perspectives.

In summary, this chapter makes the following contributions:

- We design and implement new analysis methods to systemically interpret MITRE ATT&CK evaluation's results, with evaluation dimensions including whole-graph analysis that explores the correlation capability of EDR systems and additional metrics to capture aspects of the evaluation results not covered by MITRE.
- We reconstruct several attack scenarios used in MITRE evaluation and apply whole-graph analysis to examine EDR systems' attack reconstruction and behavior correlation capabilities, which reveal whether an EDR system can effectively detect and respond to attacks.
- We propose a new evaluation metric and identify and highlight flaws in EDR systems. We also pinpoint a list of findings to shed light on areas that require improvement and offer suggestions to enhance the performance of EDR systems.

2.2 Background

2.2.1 MITRE ATT&CK Evaluation

MITRE ATT&CK Evaluation is an APT emulation conducted yearly by MITRE Corporation, started in 2018. Its participants include most leading security companies, such as Palo Alto Networks, Fortinet, and CrowdStrike. Each evaluation emulates attacks from well-known APT groups like APT3, APT29, and FIN7. Contrary to other attacks like malware and phishing, APT attacks are more complicated, involving multiple stages aiming for specific tasks. Together, those stages

form a kill chain to achieve the final goals, such as stealing sensitive information or destroying valuable properties. The MITRE Corporation has established a set of Tactics, Techniques, and Procedures (TTPs) [16] to outline each stage of the emulation process, which serve as a foundation for organizing steps in a kill chain. Tactics divide attack steps into 14 general stages, while techniques further distinguish attack steps according to the specific approach. In some cases, each technique can have associated sub-techniques, with additional details necessary to identify them accurately. Attacks performed in evaluations are illustrated step-wise, with individual steps associated with the techniques described above. Additionally, the information provided for each step in detection tests includes detection categories and modifiers, if applicable. The detection categories include the following types.

1. *Not Applicable:* The EDR system does not deploy a sensor on the given platform and thus has no visibility.

2. *None:* The EDR system deploys sensors on the given platform, but no data is available to show the event happened.

3. *Telemetry:* The EDR system knows the event happened but is unsure if they are malicious.

4. *General Behavior:* The EDR system knows the event happened and believes they are malicious. However, the system is unsure why and how the action was performed.

5. *Tactic:* The EDR system knows the event happened and believes they are malicious. The system knows why the action was performed but is unsure how the action was performed.

6. *Technique:* The EDR system knows the event happened and believes they are malicious. Additionally, the system knows why and how the action was performed.

In addition to the detection categories, modifiers provide more context about the detection. The modifiers include *delayed* and *config change*.

1. *Delayed* means the alert appears significantly late compared to the time when the attack step happens.

2. *Config change* means the alert shows up due to ad-hoc configuration modifications.

During the latest two evaluations, a new scenario was introduced to test the protection ability of EDR systems. The results of each step in the protection tests are categorized into one of the following protection categories.

1. *Not Applicable:* The EDR system does not deploy a sensor on the given platform and thus has no visibility.

2. *None:* The EDR system deploys sensors on the given platform but does not block the malicious behavior.

3. *Blocked:* The EDR system successfully blocked the malicious behavior.

To quantify the detection performance of EDR systems, MITRE defines four metrics to summarize each EDR system's capabilities at a high level: *Visibility*, *Telemetry Coverage*, *Analytic Coverage* and *Detection Count*.

1. *Telemetry Coverage* is the number of detected steps with the telemetry level detection. This is the minimum requirement for a step to be visible, as telemetry detection only confirms an event has happened but wouldn't trigger an alarm.

2. *Analytic Coverage* is the number of detected steps with some contextual information like the intention and the approach taken. Since only detection above the telemetry level is reported

as malicious behavior, the analytic coverage reflects a system's ability to detect threats from the available data.

3. *Visibility* is the number of steps with at least a telemetry detection. Note that this metric counts the number of steps in the union of *Telemetry Coverage* and *Analytic Coverage*.

4. *Detection Count* is the total number of detection made in the attack campaign. This number could be larger than the total steps, as multiple detections in different categories might be reported at a certain step.

2.2.2 Limitations of ATT&CK Evaluation

The MITRE evaluation has made significant contributions to establishing an evaluation standard for EDR solutions. However, many limitations still need to be addressed and improved upon.

2.2.2 Missing whole-graph analysis

The security field has been shifting from single-point detection to graph-based detection. The single-point detection can only detect a single step in an attack without providing an overview of the entire attack pattern. In contrast, whole graph-based detection utilizes contextual information to construct a comprehensive graph that depicts behavior and searches for threats. For modern endpoint APT defense, relying solely on single-point detection is inadequate for two reasons: Firstly, single-point detection is vulnerable to complex and sophisticated attacks that can evade traditional detection methods. Attackers can use multiple techniques to bypass single-point detection. Secondly, single-point detection focuses only on one aspect without considering contextual information, such as control and data flow, which limits perspective and could lead to false positive alarms.

Provenance graph-based detection [17], [18] overcomes these shortcomings by taking additional contextual information into account and obtaining a comprehensive view of the endpoint. Even if a single step is not identified as malicious from a single-point perspective, it can still be determined as part of the kill chain through control flow and data flow connections with other malicious behaviors. Moreover, using such correlations, malicious behaviors can be better distinguished from benign activities, reducing the number of false alarms. Most state-of-the-art endpoint detection work incorporates provenance graphs and their derivatives as part of their framework.

As discussed in the Section 2.2.1, MITRE uses a sequence of techniques to describe attack scenarios. However, the execution of kill chains in attack emulations hardly follows a linear pattern. Although such sequential representation emphasizes the detection performance on single steps, it obscures the causal and spatial aspects of the attack scenario. Fig. 2.1 shows an example of the attack graphs we constructed from an attack emulated in Wizard Spider+Sandworm (2022) evaluation. Without the graph, it's hard to understand the importance of each step in event correlation. For instance, `rundll32.exe` loads the downloaded malicious DLL file `adb.dll` to perform the following attack steps at step ⑧. Missing this step in the scope makes it hard to correlate the following attack steps with the previous setup steps. However, missing other less important steps, like `winword.exe` loading a malicious DLL file `VBDUI.DLL` at step ② does not affect the connectivity nor the causal relationship. With the help of the graph, we can investigate 1) whether the EDR systems can detect all crucial steps and 2) whether they can correlate events along the attack chain to reconstruct the attack chain and protect the system.

2.2.2 Lacking comprehensive interpretation

Another crucial part missing in MITRE evaluations is comprehensive interpretations of the results. Multiple companies claim they have achieved perfect or near-perfect performance in these

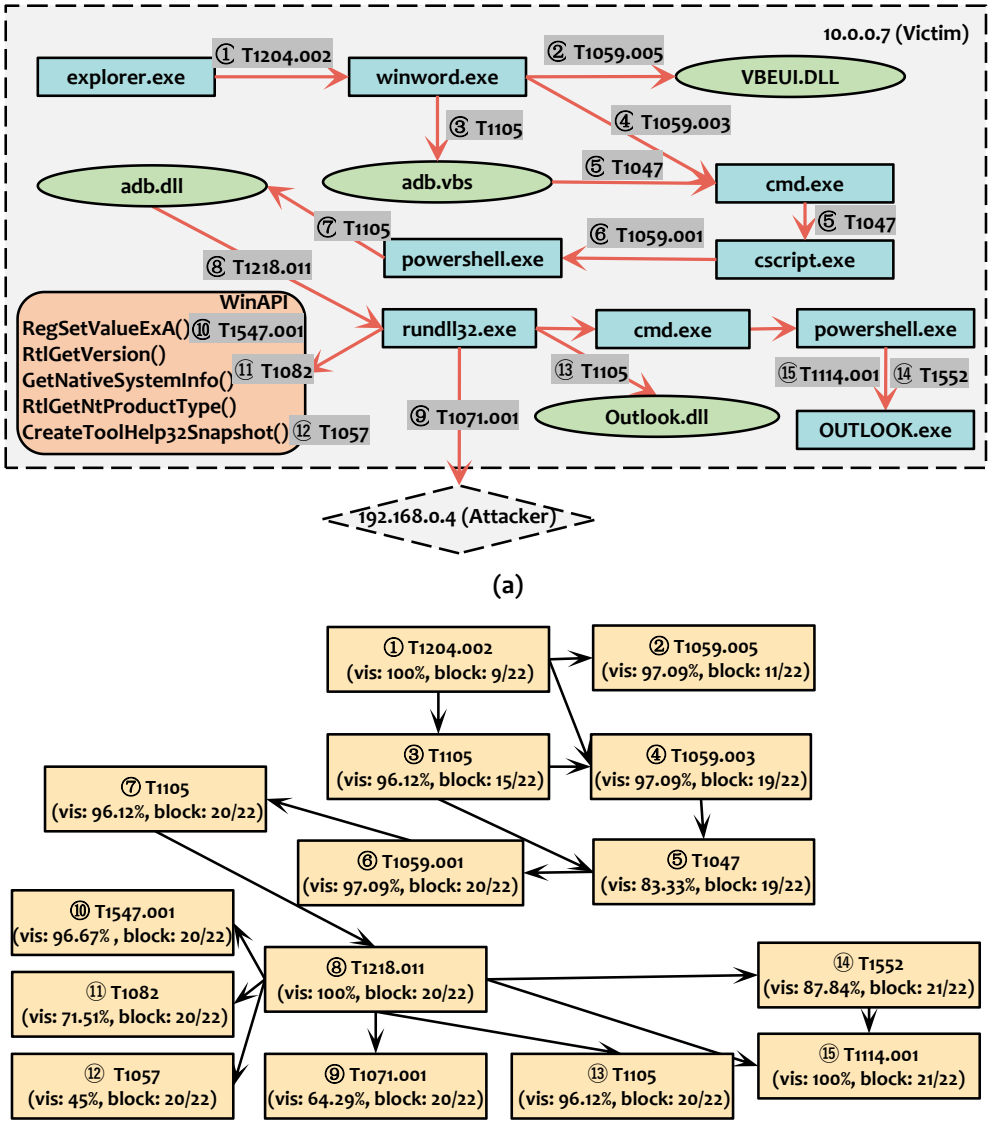


Figure 2.1: The attack graphs for scenario 1 in (Wizard Spider+Sandworm (2022) evaluation. (a) The actual attack graph. The nodes are system entities like processes and files. The edges represent system events characterized by MITRE ATT&CK techniques IDs. The numbers denote the order of events. (b) The causal relationship attack graph. The nodes are attack steps characterized by MITRE ATT&CK techniques IDs. The edges represent causal relationships between attack steps. The nodes also contain the visibility of their corresponding techniques among all EDR systems and the number of EDR systems that blocked this attack before and at this step.

evaluations. However, such claims contradict the results presented on the MITRE website. CrowdStrike [19] claims to have received 100% detection coverage across all 20 steps of the Carbanak+Fin7 evaluation. However, such a claim does not align with the evaluation results. Specifically, the '20 steps' are comprised of 174 substeps, where CrowdStrike failed to catch 22 out of 174 substeps and failed to generate alarms for 88 out of 152 visible steps. Other vendors have exhibited similar results.

Lacking clear interpretation leave room for EDR vendors to twerk the results, ironically going against the original intention of MITRE evaluation. Thus, it is necessary to add a comprehensive and objective interpretation on top of the MITRE Evaluation raw results.

2.2.2 Inconsistent evaluation framework

The MITRE Engenuity started the evaluation project in 2018. Since then, the evaluation approaches and terminology have been changing yearly, making it hard to compare the detection performance from different evaluations. For example, in the first APT3 evaluation, there are six detection categories, including *None*, *Telemetry*, *Indicator of Compromise*, *Enrichment*, *General Behavior*, and *Specific Behavior*. In the most recent Wizard Spider+Sandworm (2022) evaluation, there are five detection categories, including *None*, *Telemetry*, *General*, *Tactic*, and *Technique*. Although *None* and *Telemetry* remain the same, MITRE didn't map the rest of the detection categories. Besides, MITRE has used several versions of detection modifiers and even changed the definition of performance metrics over the years. In the most recent Wizard Spider+Sandworm (2022) evaluation, MITRE changed how detection numbers are counted. Instead of counting multiple detections on a single step, MITRE only recorded the detection with the most contextual information. In this way, visibility is the sum of telemetry and analytic coverage. The detection count metric is deleted since it is always the same as visibility.

Table 2.1: MITRE Engenuity Dataset Summary

Round of Evaluation	Participants	Steps	Techniques	# of Detection Made
APT3 (2018)	12	136	51	1970
APT29 (2019)	21	134	53	3982
Carbanak+FIN7 (2020)	29	174	46	7350
Wizard Spider+Sandworm (2022)	30	109	46	3098
Total	37	N/A	82	16.4k

Thus, in this chapter, to bridge the gap between direct results and insight, we aim to provide a comprehensive interpretation of MITRE Engenuity Evaluation results. We also try to extract the consistent aspects from the different terminology used in each year's evaluation to establish a compatible interpretation framework to compare evaluation results from different years.

2.3 Interpretation Methodology

2.3.1 Overview

We start this section by introducing the dataset in Section 2.3.2. Then, we elucidate two approaches to analyze this dataset: (1) whole-graph analysis and (2) overall statistical and trend analysis. Fig. 2.2 present an overview of our analyses. In the whole-graph analysis, we studied techniques provided in the evaluation results and APT emulation procedures published by MITRE Center for Threat-Informed Defense [20] to construct several causal relationship attack graphs. Via connectivity analysis and effectiveness analysis of the causal relationship attack graphs, we investigate the EDR systems' attack graph-level correlation and reconstruction capabilities. In the overall trend analysis, we investigate the detection performance of EDR systems on various techniques through several perspectives over the years. We aim to provide insights into the strengths and areas requiring enhancement within intrusion detection. We present the detailed methodologies in Section 2.3.3 and Section 2.3.4, respectively.

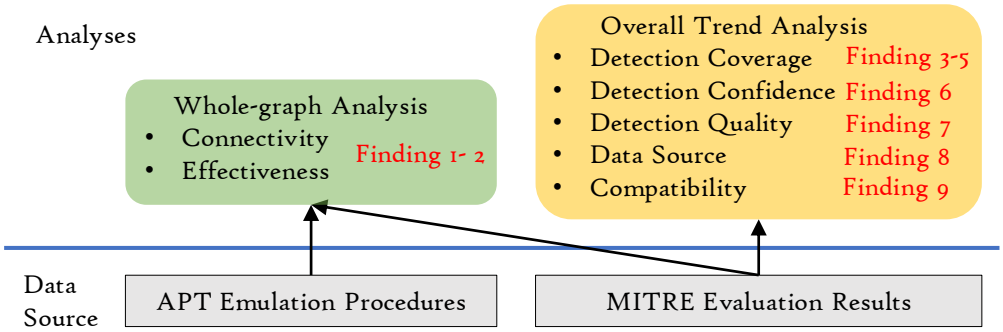


Figure 2.2: An overview of our analysis methodologies.

2.3.2 Dataset

The MITRE Engenuity has conducted four evaluations so far. Each evaluation selects one or two Advanced Persistent Threat (APT) groups and emulates several attack scenarios using the APT groups' toolkits. In total, attack scenarios consist of hundreds of steps involving dozens of techniques. For every attack scenario, MITRE Engenuity will conduct a detection test and a protection test. In detection tests, the attack kill chain will be executed without intervention. MITRE will assess whether every attack step is detected and the extent of contextual information provided during detection. While in protection tests, the defense team can intervene and stop the consequential steps in the kill chain. In this way, MITRE can assess whether an attack is contained or blocked and at which step. Therefore, the published dataset presents the results in a step-wise manner. For the detection results, data entries specify the following information about an attack step: (1) the MITRE ATT&CK technique that corresponds to the step, (2) whether the step is detected and how much contextual information is provided by the EDR system, (3) the data sources associated with detection, and (4) other miscellaneous information. The protection results also contain the MITRE ATT&CK technique corresponding to the step. Besides, instead of the detection-related information, the results show what kind of protection is triggered at each step. The definition of detection and protection categories is detailed in Section 2.2.1. Table 2.1 outlines

detailed information about the dataset associated with each campaign. Overall, we analyzed 16.4k detection results from all vendors in all published evaluation results.

2.3.3 Whole-graph Analysis

2.3.3 Attack Graph Construction

Due to the importance of provenance graph-based detection capabilities, we create a causal relationship graph model for each scenario to replace the sequential layout of MITRE evaluation results. A causal relationship attack graph is a directed graph in which the node represents an attack step in the kill chain, and the edge denotes the causal relationship between two attack steps. Constructing a causal relationship attack graph model involves nodes construction and edges construction.

Since MITRE only describes each step in terms of techniques, our initial step is to thoroughly examine each step's corresponding procedure, which is subsequently classified into descriptive and causal categories. The descriptive techniques solely describe specific features of a step without establishing any causal connections with other entities (e.g., *Encrypted Channel*). In contrast, the causal techniques interact with other entities, such as creating a process or writing to a file, thereby establishing causal relationships with other steps (e.g., *Ingress Tool Transfer*). All steps corresponding to causal techniques become the nodes in our causal relationship attack graph.

After classifying each step, we examine the subject and object of each step to establish causal relationships. When we connect the subject to the object, the edges are constructed in the graph. We generally consider two kinds of causal relationships: control flow and data flow. Firstly, the control flow creates causal relationships via process creation. If an attack step is performed by a process created in a previous step, then the two steps establish a control flow causal relationship. Secondly, the data flow creates causal relationships via communication over files. If an attack

step reads a file written in a previous step, the two steps establish a data flow causal relationship. Fig. 2.1(b) and 2.5(b) in the Appendix show two examples of causal relationship attack graphs.

2.3.3 Attack Graph Analysis

After constructing a causal relationship attack graph, we analyze EDR systems' detection and protection performance from the attack graph perspective. We aim to answer the following two questions: (1) whether the EDR systems can fully reconstruct the attack kill chain; (2) whether the EDR systems can effectively aggregate behaviors along the kill chain to understand its severity. We answer the first question by conducting a connectivity analysis on the causal relationship attack graphs constructed from the detection results. As for the second question, we analyze the protection performance to examine the EDR systems' effectiveness. An effective EDR system should detect and respond to threats at the appropriate time. We study several attack cases and investigate when EDR systems block the kill chain.

Connectivity Analysis: We examine the kill chain visibility by counting the connected components in the causal relationship attack graph and comparing them with the ground truth. Suppose a campaign involves attacks on three individual hosts, leading to three separate kill chains among those hosts. If the number of connected components on the graph is more than three, one or more kill chains are broken into multiple small segments. If the number is less than three, at least one kill chain is completely missing in the detection. If the number equals three, we check if the three segments match the ground truth. In this way, we use the connected components as a metric to evaluate EDR systems' attack reconstruction capability.

Effectiveness Analysis: We examine which step a blockage is triggered given each vendor's detection results to analyze how effectively the EDR systems use the graph information. We assume EDR systems are knowledgeable about the maliciousness of different behaviors, and they would

block the attack once the severity of existing behaviors accumulates to a certain threshold. We select a few attacks to perform case studies. For each case, we manually determine a step on the attack kill chain when the malicious intention is evident as the baseline. Then, for each EDR system, we compare at which step the attack is blocked with the baseline. If the attack is blocked earlier than the baseline, it suggests the EDR system adopts an aggressive strategy in defense response. In this case, benign behaviors could be incorrectly classified as malicious, leading to unpredictable problems. If the attack is blocked later than the baseline, it suggests the EDR system cannot react to threats in time. Such delay could allow the attacks to happen unhindered.

2.3.4 Overall Trend Analysis

Besides analyzing the evaluation results on the attack graph, we also investigate the detection and protection performance of individual techniques and EDR systems over the years.

2.3.4 Detection Coverage

MITRE presents the evaluation from the vendors' perspective. Each vendor's performance is reflected by its analytic coverage, telemetry coverage, and visibility on all attack steps. We want to investigate EDR systems' performance from another perspective: how well all vendors can detect each technique. We analyze the detection coverage of a technique from two perspectives: visibility and analytic coverage. Like MITRE's metrics, we determine the visibility of a technique by calculating the ratio of EDR systems aware of the corresponding behaviors. We determine the analytic coverage of a technique by calculating the percentage of the vendors that successfully detected the corresponding step.

1. *Visibility*

$$V(x) = \frac{S_v(x)}{S_t(x)}$$

Where x is a specific EDR system or a specific technique. $V(x)$ is the visibility score of the given x. $S_v(x)$ is the number of visible substeps of the given x, and $S_t(x)$ is the total substeps related to the given x.

2. *Analytic Coverage*

$$A(x) = \frac{S_a(x)}{S_v(x)}$$

Where $A(x)$ is the analytics score of the given x, $S_a(x)$ is the number of detections beyond the telemetry level of the given x, and $S_v(x)$ is the number of visible substeps of the given x.

2.3.4 Detection Confidence

Although the three metrics adopted from MITRE evaluation provide helpful information, some aspects are missing. Specifically, those metrics do not take different detection categories into account. In other words, a system reporting all malicious behaviors in the general behavior level would receive identical scores as a system reporting all malicious behaviors in the technique level. We propose an additional metric: *confidence* to address this issue.

Confidence is a weighted score calculated by multiplying the percentage of detection made from different detection methods by the corresponding weight multiplier. A high confidence score suggests more details about the malicious behavior are provided.

$$C(x) = \frac{4D_{te}(x) + 3D_{ta}(x) + 2D_{ge}(x) + D_{tel}(x)}{4D_v(x)}$$

Where $C(x)$ is the confidence score of a given technique or EDR system x. $D_{te}(x)$ is the number of technique detection of the given x, D_{ta} the number of tactic detection of the given x, D_{ge} the number of general behavior detection of the given x, D_{tel} the number of telemetry detection of the given x, and D_v the total visible substeps of the given x. The multiplier associated with each

variable indicates the granularity of the detection results, with four being the most detailed and one being the most general. Since MITRE evaluation provides four levels of granularity for detection results, we intuitively use 1 through 4 as the multipliers. They can be further adjusted if more detailed data is available.

2.3.4 Detection Quality

Another aspect missing from MITRE-provided metrics is the negative modifiers. A system reporting all alarms with significant delay or configuration change receives identical scores as a system reporting all alarms with no delay and no configuration change. To examine the presence of modifiers quantitatively, we propose a *quality* metric for techniques and EDR systems. For a technique, the quality score is the ratio of visible substeps without negative modifiers to the total visible substeps. A high-quality score implies low detection latency and adequate out-of-box usability.

$$Q(x) = \frac{S_m(x)}{S_v(x)}$$

Where $S_m(x)$ is the number of visible substeps without negative modifiers of a given technique or EDR system x, and $S_v(x)$ is the total visible substeps of the given x. We treat all the negative modifiers equally since they are all related to manual adjustments or analyses.

2.3.4 Data Source

Besides the quantitative analysis specified above, we investigate the data sources used in evaluations via a rather qualitative approach. Specifically, we compare the data sources used in each year's evaluation to examine the scope of data sources used in EDR systems. We also discuss the frequency of a data source used in each evaluation to investigate the importance of the data source.

2.3.4 Compatibility

We investigate EDR systems' compatibility from two perspectives: availability and performance. We examine the availability by calculating the ratio of EDR systems that support a given platform. Since MITRE Engenuity evaluations only involved Windows and Linux platforms so far, we will focus on the availability of these two platforms. Besides, we compare the detection performance on different platforms.

2.4 Whole-graph Analysis

Since whole-graph analysis is specific to the attack scenarios, we use attack scenarios in the Wizard Spider+Sandworm (2022) evaluation as the cases to perform our whole-graph analysis. We constructed casual relationship attack graphs at the procedure level for all attack scenarios in Wizard Spider+Sandworm Evaluation. Fig. 2.1 and 2.5 in the Appendix present two examples of constructing the causal relationship attack graphs.

2.4.1 Connectivity Analysis

We analyze the attack graph connectivity by calculating the number of connected components in the casual relationship attack graph generated from the visible steps of each vendor and comparing it with the ground truth. There are six hosts involved throughout the attack emulation. Thus, there should be six segments. One of the six hosts runs under the Linux environment, and the other five are under the Windows environment. 22 vendors support Linux environment data collection and detection out of 30 participants. Therefore, we divide the vendors into two groups according to their Linux platform compatibility. Of the 22 vendors that support the Linux platform, three have more than six segments (Rapid7 has 13, Cisco has 10, and Cylance has 11). Of the eight vendors

Table 2.2: Summary of Protection Test Results

Test	# of Blockage	# of Participants	Protection Rate
1	21	22	95.5%
2	21	22	95.5%
3	16	22	72.7%
4	12	22	54.5%
5	15	22	68.2%
6	20	22	90.9%
7	9	17	52.9%
8	20	22	90.9%
9	18	22	81.8%

that don't support the Linux platform, two have more than five segments (Deep Instinct has nine, and ReaQta has six). 25 out of 30 (83.3%) participants can obtain a visibly connected subgraph containing all attack steps. Thus, we conclude that most vendors can see the connection between attack steps on a graph level.

2.4.2 Effectiveness Analysis

We analyze all protection evaluation scenarios and discuss two as case studies to see how effectively the EDR systems use the graph information. Table 2.2 summarizes the results of nine protection tests. 22 vendors participated in the protection tests. Test 7 is conducted on Linux. Since five participants didn't support the Linux platform, only 17 were in Test 7. Fig. 2.1 and 2.5 in the appendix presents the two cases we will discuss in detail.

Scenario 1: Emotet Initial Compromise, Persistence, and Collection.

Fig. 2.1 shows a detailed attack graph of this scenario. In this scenario, the adversary sent a Word document over email, which contained obfuscated VBA macros that downloaded and executed a malicious DLL based on the malware Emotet. The malicious DLL then established a command and control (C&C) session with the adversary server. Besides, it achieved persistence by modifying the registry via the WinAPI function `RegSetValueExA()`. Later, the malicious DLL collected process information by calling the WinAPI functions `CreateToolhelp32Snapshot()`

and `Process32First()`. Finally, it downloaded another malicious DLL to search for credentials in Outlook.

21 out of 22 EDR systems blocked this attack at different steps. Most blockages happen at the first step when the Explorer executes the Word document. This behavior is already pretty suspicious, as it downloaded an untrusted file. In the following steps, Word downloaded a malicious DLL file and a malicious VBS file and then executed it. The malicious intention is evident at this step, as this is a typical download and execution behavior. However, not all EDR systems responded to it: 19 EDR systems blocked the process, while three EDR systems either waited until later to block or didn't react. We checked their connectivity and found they could see the connected attack steps corresponding to this protection test. Although they have reasonable detection performance on single steps, these EDR systems failed to chain steps together to better understand the kill chain to provide appropriate protection.

Scenario 2: TrickBot Execution, Discovery, and Kerberoasting.

Fig. 2.5 in the Appendix shows a detailed attack graph of this scenario. In this scenario, the adversary authenticated into the victim's host using stolen credentials from scenario 1. Then, the adversary downloaded and executed a malicious EXE derived from TrickBot. The malicious EXE first established a C&C session with an adversary-controlled server. Then, it collected various system information by executing shell commands. Finally, it downloaded a tool called `rubeus` to perform Kerberoasting [21], which could steal encrypted credentials.

21 out of 22 EDR systems blocked this attack at different steps. In this scenario, the adversary connected to the target via RDP protocol as the first step. Looking at this step alone, it could be normal behavior. However, in the latter steps, the adversary downloaded a malicious file and executed it to establish a communication channel with the C&C center. The malicious intention is evident at this step as it downloaded and executed an unknown file and established a suspicious

outward connection. Only half of the EDR systems decided to block the process at steps 3 and 4. The rest of the EDR systems block the process when collecting system information in the later steps.

Although scenarios 1 and 2 had the same protection rate eventually, there is a noticeable delay in scenario 2 compared to scenario 1. One reason could be the difference in step visibility. As shown in Fig. 2.1(b) and 2.5(b), early steps in scenario 1 had better visibility than early steps in scenario 2. The third step in scenario 2 only had 64.29% visibility, making it hard for EDR systems to gather enough information and react.

Finding 1: Attack graph level correlation capabilities are necessary to achieve good defense because isolated single steps cannot provide enough confidence for EDR systems to respond.

The steps in Tests 3, 4, 5, and 6 happened as a connected kill chain on the same host in the detection test but are isolated into different test scenarios in the protection tests. This gives us a chance to investigate how isolated scenarios can affect defense. Since isolated scenarios contain fewer steps for correlating, the defense and response decisions primarily rely on single-step detection. We observed a significant drop in protection rate in tests 3, 4, and 5 as shown in Table 2.2. Test 4 only contained two steps that dumped system information (C disk and the registry) and received the lowest protection rate. Although dumping the entire C disk and the registry seems suspicious, such behaviors alone are usually not malicious enough to be escalated to alarms. Admittedly, we observed many cases in which EDR systems take action as soon as a suspicious file is downloaded and executed. Still, such a download and execution pattern wouldn't work well against file-less attacks, living-off-the-land attacks, and other evasion techniques.

Furthermore, some evidence shows EDR systems with poor performance didn't have graph-level correlation capabilities. For example, the detection screenshots from vendors like Deep Instinct didn't present any graph-level information along with the detection. In contrast, the detection

screenshots from vendors like Sentinel One complement the detection with kill chain information on a graph.

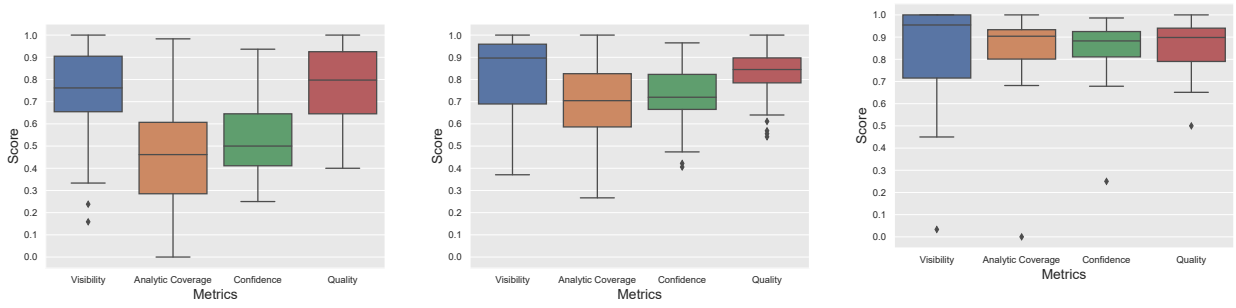
Finding 2: Although some EDR systems demonstrated good attack graph level correlation capabilities, we still identified three practical problems: delay in protection, lack of protection, and lack of cross-host correlation capability.

Delay in protection problem exists ubiquitously in all scenarios. In the cases we analyzed, the adversary will log into the target and download a payload. More than half of the protection happens here, but the rest would happen either after the adversary had done some malicious behaviors or not at all. We checked their visibility. Most of them can see a connected attack chain. Those EDR systems require a longer kill chain to accumulate confidence before blocking a process. Such a mechanism prevents them from reacting quickly to threats. Sometimes, this even prevents them from reacting at all.

Lack of protection problems would still occur when some stealthy steps are applied. Besides requiring a longer chain to reach their confidence level, some EDR systems are susceptible to attack evasion techniques. For example, Test 3 modified the registry to achieve persistence. Tests 4 and 5 mimic system administrators to dump system information and modify system configurations. These tests applied more stealthy and sophisticated approaches than other tests, thus receiving a relatively low protection rate.

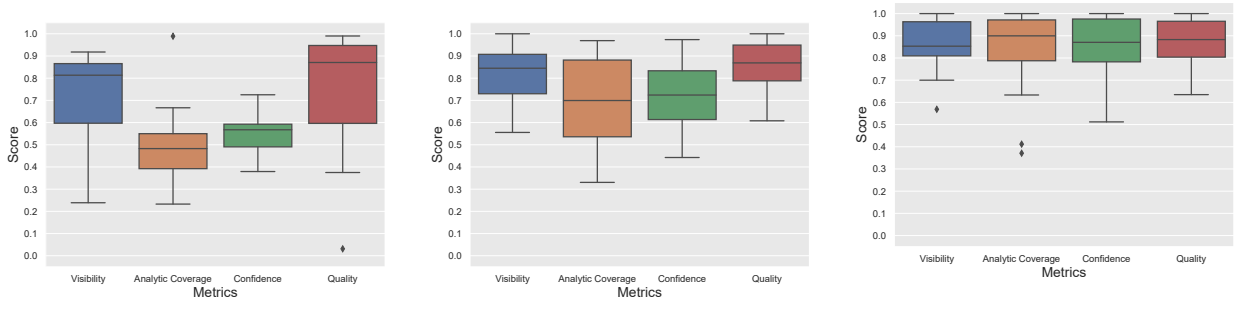
Attack graph level correlation should be applied on individual hosts and across hosts. In this evaluation, the adversary used the same tools and adopted similar attack patterns on different hosts in tests 2 and 3, respectively. Given test 2 happened before test 3, the protection performance in test 3 is not better than test 2. It suggests the EDR systems could not learn from the happened attacks to react to similar attacks in the future. Furthermore, no evidence shows that detection and response mechanisms use information across hosts to improve defensive performance.

2.5 Overall Trend Analysis



(a) APT29 (2019) (b) Carbanak+Fin7 (2020) (c) Wizard Spider+Sandworm (2022)

Figure 2.3: Technique perspective score distribution of each metric in different evaluations. The metrics are visibility (blue), analytic coverage (orange), confidence (green), and quality (red) from left to right, respectively.



(a) APT29 (2019) (b) Carbanak+Fin7 (2020) (c) Wizard Spider+Sandworm (2022)

Figure 2.4: Vendor perspective score distribution of each metric in different evaluations. The metrics are visibility (blue), analytic coverage (orange), confidence (green), and quality (red) from left to right, respectively.

In this section, we examine several perspectives in all available datasets from MITRE Engenuity to investigate the paradigm of attacks and defenses in a real-world setting. We mainly analyze the results from more recent evaluations, especially the Carbanak+Fin7 (2020) and the Wizard Spider+Sandworm (2022) evaluations because they included APT emulations performed on multiple operating systems and they used a more established taxonomy to describe evaluation results compared to the previous evaluations.

2.5.1 Detection Coverage

We calculate the visibility and analytic coverage scores for individual techniques across the EDR systems participating in the evaluations. The distribution of visibility and analytic coverage scores from the technique perspective are shown in Fig. 2.3. We also calculate the visibility and analytic coverage scores for individual EDR systems as shown in Fig. 2.4. Then, we use these two metrics to analyze the overall trend of detection coverage. Moreover, we select a few EDR systems and techniques that receive excellent or very low scores and try to analyze the reasons behind them.

2.5.1 Visibility

As shown in Fig. 2.3 and 2.4, the distributions of technique visibility scores and vendor visibility scores are mostly skewed to the left. The median of technique visibility distribution from Wizard Spider+Sandworm (2022) evaluation is around 95%, which means half of the attack steps can be seen by at least 95% of the EDR systems in evaluations. The median vendor visibility distribution from the same evaluation is around 85%, suggesting half of the EDR systems can see more than 85% of the attack steps. Comparing the visibility score distribution in the three evaluations, we see an obvious improvement in the median and the lowest score for both techniques and vendors over the years.

Two techniques in Command and Control (C&C) receive low scores across the EDR systems in the Carbanak+Fin7 evaluation. Specifically, only around 40% of the EDR systems can record *Encrypted Channel* or *Application Layer Protocol* communications. In this evaluation, such communications include transmitting data over SSH protocol, MSSQL transactions, and HTTPS protocol. Meanwhile, the visibility of other C&C techniques, which establish the connection via TCP, are all above 80% among the EDR systems. Based on this observation, we conclude most EDR systems selectively collect network traffic data on the transport layer and ignore the application

layer protocols.

Surprisingly, 15 techniques achieved perfect coverage among all EDR systems in the Wizard Spider+Sandworm evaluation, including six discovery techniques, three Defensive Evasion techniques, and a few techniques in other tactics. Those techniques involve manipulating or investigating system configurations and services, which implies all EDR systems have no trouble monitoring system configurations and services. In addition, four out of six techniques in the Execution Tactic receive a visibility score above 90% in the Carbanak+Fin7 evaluation. Those techniques involve a process loading certain libraries or executing specific commands. We conclude that all EDR systems emphasize monitoring process loading and execution so that the techniques related to execution achieve the best visibility among all techniques.

SentinelOne achieves 100% visibility on all techniques used in the Carbanak+Fin7 evaluation, while only around 50% of the techniques are visible to AhnLab in the same evaluation. AhnLab hardly monitors the file system, as most of the techniques associated with *Collection* and *Credential Access* remain invisible to AhnLab. Furthermore, AhnLab does not monitor network activities except for a few file download behaviors and some TCP connections. It is particularly interesting that AhnLab can see and even raise alarms on some file download behaviors by Powershell but remain completely blind to the rest of the file download behaviors by Powershell from the same IP.

We observe changes and continuities by comparing the visibility score between the Carbanak+Fin7 and Wizard Spider+Sandworm evaluations. Most EDR systems obtain higher visibility scores in the latter evaluation, especially AhnLab, whose visibility score has a huge boost from 0.517 to 0.761. This implies the entire industry has improved in making various behaviors visible. The visibility difference in techniques is interesting. The two evaluations have about the same average visibility scores (about 80%). However, some techniques like *Archive Collected Data* have a perfect visibility score in the Carbanak+Fin7 evaluation but only receives a 0.033 visibility score

in the Wizard Spider+Sandworm evaluation, which means it's only visible to one out of 30 EDR systems. The dramatic discrepancy suggests technique is not an appropriate unit for detection coverage since the same technique could be implemented with totally different procedures and consequently require distinct detection capabilities.

Finding 3: Most EDR systems have good data collection capability, and this capability is improving every year. In the most recent Wizard Spider+Sandworm (2022) evaluation, 75% of the EDR systems can identify more than 80% of the attack steps.

Finding 4: Large discrepancies in visibility for the same technique in different evaluations suggest that techniques are still too coarse-grained for detection coverage. A more fine-grained unit, such as generalized technique implementations, is needed.

2.5.1 Analytic Coverage

As shown in Fig. 2.3 and 2.4, analytic coverage has significantly improved over the years. In the most recent Wizard Spider+Sandworm (2022) evaluation, 50% of visible attack steps would trigger alarms on more than 90% of the EDR systems, and 50% of the EDR systems would generate alarms for at least 90% of the visible attack steps. Unlike the distribution of visibility, analytic coverage exhibits a nearly symmetric distribution.

The techniques receiving low analytic scores are mainly in C&C, Exfiltration, and Collection tactics. Not surprisingly, *Archive Collected Data* has an analytic coverage score of 0 given its low visibility score, which means only telemetry detection is made for this technique in the Wizard Spider+Sandworm evaluation. Other techniques like *Exfiltration Over C&C Channel* and several C&C techniques also received below 50% analytic coverage scores. Although such behaviors might be indistinguishable from everyday benign behaviors, they could still be identified as a part of a complete attack chain. Thus, detecting these behaviors challenges EDR systems' capabilities

of assembling attack chains from scattered individual events. Low analytic coverage scores on those techniques suggest all EDR systems have room for improvement in their event correlation capability.

Moreover, the techniques receiving high analytic coverage scores mostly fall into categories of Defense Evasion and Credential Access, including *OS Credential Dumping*, *Inhibit System Recovery* and *Process Injection*. Such techniques are easy to separate from normal behaviors as they carry malicious intentions conspicuously.

Comparing the analytic coverage scores between the Carbanak+Fin7 and Wizard Spider+Sandworm evaluations, the analytic coverage scores have a significant improvement for all techniques as the average increases from 69.8% to 85.5% and the standard deviation decreases from 16% to 8%. Techniques in C&C, Exfiltration, and Collection tactics that receive less than 60% analytic coverage scores in the Carbanak+Fin7 evaluation mostly have higher than 70% analytic coverage scores in the Wizard Spider+Sandworm evaluation.

Finding 5: Although EDR systems' ability to determine malicious behaviors improves over time, they struggle to detect 'living-off-the-land' threats. Throughout all evaluations, several techniques, including *Encrypted Channel*, *Exfiltration Over C&C Channel*, and *Archive Collected Data*, etc., consistently had worse detection rates. It is hard for EDR systems without event correlation capabilities to discern whether such individual steps are malicious at the system provenance level.

Moreover, we divide EDR systems and techniques into four quadrants according to their visibility and analytic coverage scores for comparison. As shown in Fig. 2.6 and 2.7 in the Appendix.

For EDR systems, the top performers like Sentinel One and Palo Alto Networks at the top right corner are excellent at visibility and detection, and the bottom performers like AhnLab at the bottom left corner need to catch up with both capabilities. Interestingly, there are some ramifications

among the EDR systems in the middle. The five EDR systems in the top left quadrant, like FireEye, tend to focus on detection capability. FireEye has way above average analytics score (90.3% vs. 69.8%) but slightly below average visibility score (78.2% vs. 80.8%). On the contrary, the six EDR systems in the bottom right quadrant, like CrowdStrike, tend to put more emphasis on visibility capability. CrowdStrike has a higher-than-average visibility score (87.4% vs. 80.8%) but a lower-than-average analytic coverage score (46.2% vs. 69.8%).

For techniques, 42 out of 63 falls into the top right quadrant and the bottom left quadrant, meaning they have comparable visibility and analytic coverage scores. For instance, *Network Share Discovery* has an excellent visibility score (100%) and analytic coverage score (90.1%), while *Exfiltration Over C&C Channel* has a poor visibility score (43.7%) and analytic coverage score (22.6%). Six out of 63 techniques fall into the top left quadrant. Those techniques receive below-average visibility scores but above-average analytic coverage scores. It implies it is hard for EDR systems to collect related data, but they can be easily identified as malicious behaviors once visible. For example, *Access Token Manipulation* receives only a 42.9% visibility score but an 80.6% analytic coverage score. Such techniques often involve some defense evasion intentions, or it might be expensive to collect related data, making them naturally stealthy. 15 out of 63 techniques fall into the bottom right quadrant. Those techniques receive above-average visibility scores but below-average analytic coverage scores. It suggests they are easily visible to EDR systems, but it is hard to associate them with malicious behaviors. For example, *Email Collection* has a 95.2% visibility score but only a 37.5% analytic coverage score. Such behaviors are not stealthy but blend in with benign behaviors a normal user would perform.

To sum up, 18 out of 29 EDR systems have comparable visibility and analytic coverage scores. In comparison, five of the rest focus more on their detection capability, and six EDR systems lean towards improving their visibility. 42 out of 63 techniques have comparable visibility scores and

analytic coverage scores. Among the remaining 21 techniques, six techniques are hardly visible but easy to detect once they are visible. The other 15 techniques are easily visible but hard to identify as malicious behaviors.

2.5.2 Detection Confidence

The layout of confidence scores looks similar to analytic coverage scores since they reflect the soundness of detection results. However, we propose the confidence metric in addition to the analytic coverage metric because the confidence metric takes different detection levels above telemetry into account. As a weighted average, the confidence score indicates the overall detection level made by an EDR system or against a technique. Along the spectrum of confidence scores, a 25% confidence score means the detection level is at telemetry on average; a 50% confidence score means the detection is at General Behavior on average; a 75% and 100% confidence score means the detection level is at Tactic and Technique, respectively. For instance, *Input Capture* has a perfect analytic coverage score, while it only has a 61.7% confidence score in Carbanak+Fin7 (2020) evaluation, suggesting most of the detection levels are between General Behavior and Tactic. Other Credential Access techniques like *Unsecured Credentials and Credentials from Password Stores* also have discrepancies in analytic coverage and confidence.

Interestingly, the techniques that receive the highest and lowest confidence scores are all in Credential Access or Discovery tactics. *OS Credential Dumping* and *Network Share Discovery* receive above 90% confidence scores, whereas *Credentials from Password Stores* and *Permission Group Discovery* obtain below 30% confidence scores. All EDR systems can identify malicious behaviors related to *OS Credential Dumping* and *Network Share Discovery* and complement alarms with additional context like motivations and techniques used. On the contrary, EDR systems struggle to set off alarms for malicious behaviors related to *Credentials from Password Stores* and *Permission*

Group Discovery, let alone providing additional context.

The technique *Web Service* has a significantly lower confidence score than its analytic coverage score. This suggests the alarms on *Web Service* fail to provide detailed contexts, like the motivation of such behavior or the specific technique used. Such vague alarms usually require system administrators to spend more time investigating and responding. On the other hand, techniques like *Exfiltration Over C&C Channel* remarkably higher confidence scores than its analytic coverage score, which implies detection on *Exfiltration Over C&C Channel* comes with a satisfying amount of details although the number of generated alarms is relatively low.

For EDR systems, the confidence scores and analytic coverage scores also share the general trend but with a few outliers. We calculate the confidence-analytic difference across all EDR systems participating in Carbanak+Fin7 (2020) evaluation to further investigate their detection confidence. CyCraft has a notably lower confidence score than the analytic coverage score, suggesting CyCraft puts more emphasis on detection coverage than other contexts. On the contrary, EDR systems like Sophos and AhnLab have much higher confidence scores than their analytic coverage score, which suggests they value the context provided in detection more than the coverage. Top performers like Palo Alto Networks, Sentinel One, and CheckPoint have about the same confidence score and analytic coverage score, which implies their detection has satisfying coverage and abundant details.

Comparing the distribution of confidence scores over the three years in Fig. 2.3 and 2.4, the technique median confidence score improved from 50% in APT29 to 72% in Carbanak+Fin7, and finally to 88.25% in Wizard Spider+Sandworm. The confidence score distributions exhibited a decrease in range while shifting towards the higher end. This implies a significant enhancement in the level of detail with the detection. In the APT29 (2019) evaluation, only 50% of the attack steps can be detected at the General Behavior level or above by all EDR systems on average; however,

Table 2.3: Data Sources in MITRE Evaluations

Campaign	# of Data Source	Top 3 Data Sources
APT29 (2019)	9	File, Command, Process
Carbanak+FIN7 (2020)	25	Process, File, Network Traffic
Wizard Spider+Sandworm (2022)	41	Process, File, Network Traffic

in the Wizard Spider+Sandworm (2022) evaluation, 75% of the attack steps can be detected at the Technique level on average.

Finding 6: Different amounts of details in alarms reflect EDR systems' detection confidence. Throughout the five evaluations, EDR systems are less confident to trigger alarms on techniques that widely exist in everyday activities such as *Email Collection*, *Exfiltration Over C&C Channel*, and *Ingress Tool Transfer*. Thus, EDR systems tend to provide less contextual information, like their roles in the attack chain. On the contrary, EDR systems are more confident in detecting typical malicious behaviors like *OS Credential Dumping* and *Network Sniffing*. EDR systems' overall detection confidence has improved remarkably, as shown in Fig. 2.3 and 2.4.

2.5.3 Detection Quality

We calculated the quality metric for techniques and EDR systems in the recent three evaluations, as shown in Fig. 2.3 and 2.4, respectively. The detection quality score has been increasing over the years. *Credentials from Password Stores* receive the lowest quality score (54.2%) in the Carbanak+Fin7 evaluation, which suggests significant delay and manual efforts are involved. In this scenario, the credentials stored in the Chrome web browser are accessed via a malicious tool. This technique has a fairly low Visibility score and analytic coverage score. Even when it is detected, it usually requires human analysis. Other techniques with low quality scores (below 60%) include *Inter-Process Communication*, *Credentials from Password Store* and *Data from Local System*, which require modifying detection policies depending on the local environments. We suspect

failures in detecting this technique are also related to systems' weak ability to link individual events to an attack chain. While accessing the credentials stored in the browser itself doesn't seem very suspicious, downloading an unknown tool and using it to access credentials makes it very suspicious.

OpenText and DeepInstinct receive the lowest quality score (around 60%) among the EDR systems, while some EDR systems like SentinelOne, ReaQta, and CyCraft obtain close perfect quality scores in the Carbanak+Fin7 evaluation. The score differences imply EDR systems' various self-adapting abilities. Systems with high quality scores can work effectively in different environments without much human intervention, while systems with low quality scores require a lot of manual tuning and analysis.

Finding 7: EDR systems often require extra manual effort to detect techniques that are closely integrated with local environments, such as *Credentials from Password Stores* and *Inter-Process Communication*. Only four tested systems in the Wizard Spider+Sandworm (2022) evaluation do not require extra effort to detect such techniques.

2.5.4 Data Source

The number of distinct data sources has been changing over the years. No data source information is available in the APT3 evaluation (2018). Since the APT29 evaluation (2019), MITRE has started to collect data source information in the detection results. As shown in Table 2.3, nine distinct data sources are recorded in the APT29 evaluation results, whereas in the most recent Wizard Spider+Sandworm evaluation (2022), 41 different data sources are recorded. As the number of distinct data sources increases over the years, not only do the existing data sources become more specific, but some new data sources are also included in the data sources. At the same time, the taxonomy of data sources has been changing. In the APT29 and Wizard Spider+Sandworm eval-

uations, the data sources are recorded in *category: sub-category* format. In contrast, in the Carbanak+FIN7 evaluation, the data sources are recorded as *category* without further sub-categories. In the APT29 evaluation, the data sources are from the process, file, registry key, and network connection creations, as well as script and command line executions. In the Wizard Spider+Sandworm evaluation, network-related data sources include network connection creation, traffic content, and traffic flow. Besides, additional data sources, such as firewall metadata and network share access, are included. Our findings in Section 2.5.1 demonstrated such enrichment in data sources has a positive correlation with the improvement in the detection performance over the years.

Finding 8: Increasing complexity and variety of data sources suggest EDR systems can utilize extensive information from different dimensions. Although an increasing number of data sources are used, the top data sources remain unchanged. Process, file, network, scripts, and system calls/APIs are still the most fundamental and valuable data sources for EDR systems.

2.5.5 Compatibility

In the first two evaluations, APT3 and APT29, the target environments only contain Windows hosts; thus, all participants must support the Windows platform. Starting from the Carbanak+Fin7 evaluation in 2020, MITRE enrich the variety of target environments by including Linux servers as parts of the target system. In the Carbanak+Fin7 evaluation, 22 out of 29 participants supported the Linux platform. In the following Wizard Spider+Sandworm evaluation, 22 out of 30 participants supported the Linux platform.

In Section 2.4.2, we found EDR systems had a low protection rate against attacks on Linux. The attack on the Linux platform follows a similar pattern to the ones on Windows: uploading a payload and using it to establish C&C connections. Given the attacks have similar visibility and attack pattern, most vendors can protect against the attacks on Windows, but the attacks on Linux

were only blocked by around half of the EDR systems.

Finding 9: Data collection and protection capability needs improvement on the Linux platform since around 25% of the evaluated EDR products don't support the Linux platform. For similar attack patterns, EDR products present worse protection results on Linux than on Windows.

2.6 Related Work

2.6.1 Endpoint Detection and Response (EDR)

An increasing number of researches on endpoint security solutions have been conducted to improve APT defense methods and forensics technologies on various platforms. EDR frameworks like HOLMES [22], Poirot [23], MORSE[24], and others [25]–[35] aim to improve defense on Windows and Linux operating system, while other methods like RiskRanker [36] and E-EMD [37] target security on mobile and cloud platforms, respectively. However, they all use statistical measurements like false positive and true positive, precision, recall, accuracy, and F-Score, to describe the detection performance. They also include CPU and memory usage as measurements for overhead. However, it is hard to compare the measurements from different works due to the different datasets and hardware used to carry out the measurements. Surveys on the EDR systems [38]–[40] mainly focus on the methodology and dataset used but pay little attention to EDR evaluation.

In efforts to improve security evaluations, researchers have been studying the evaluation flaws in existing security works. For instance, Van Der Kouwe et al.[41] identifies a list of common benchmarking mistakes and indicates that benchmarking flaws exist widely in system security papers published in top venues, which suggests the necessity of standardizing security benchmarks. Following those papers' insights and suggestions, we propose our evaluation and interpretation framework for system security work in academia.

2.6.2 Security Benchmark

Most existing works focus on generating representative data sets since quality security data sets are scarce. As early as 1998, DARPA launched its intrusion detection evaluation [8] in collaboration with Lincoln Lab. Zuech et al. [9] tried to generate network-based data sets for evaluating network intrusion detection systems (NIDS); Divekar et al. [10] modified existing network-based data sets to improve training performance in anomaly-based NIDS; Almakhdhub et al. [11] targets benchmarking Internet of Things (IoT) devices. Additionally, the data set from the DARPA Transparent Computing program [12] has been used widely in recent security work. Still, data from only two out of five attack campaigns are publicly available, and thus the attack scenarios are minimal. Although such data sets help mitigate the deficit in security evaluation data, they do not provide extensive methodologies for interpreting the results.

Some other work aims to improve the explainability of evaluations. Hao et al. [13] and Mendes et al. [14] designed methodologies to obtain more explainable evaluation results for static application security testing (SAST) tools and web serving systems, respectively. However, their methods are specific to the targeting tools or systems. Thus, it is hard to expand the methodologies to other security fields.

Recently, some security studies have used the knowledge base built by MITRE ATT&CK. Choi et al. [42] used the tactic, technique, and procedure (TTP) proposed by MITRE ATT&CK to generate attack sequences. On the other hand, Outkin et al. [43] use attacks emulated in MITRE ATT&CK evaluation as the attack models to discuss defender policy and resource allocation. Although they used MITRE ATT&CK knowledge base, they didn't analyze and interpret MITRE ATT&CK evaluations.

Other commercial security 'benchmarks' apply miscellaneous self-designed metrics in various testing environments, purely focusing on comparing EDR vendors on behalf of the customers for

marketing purposes. For instance, Gartner tries to address the benchmarking challenges with its Magic Quadrant [6]. However, the methodology is not transparent and is lack of explanation. Moreover, Gartner emphasizes secondary concerns for businesses like value and viability, which don't provide insights on improving the security systems' performance. Another attempt in the industry is AV-Comparatives [7], which evaluates the anti-virus capabilities of security products. However, the evaluation methodology is not transparent like Magic Quadrant's, and the evaluations only focus on a narrow range of attack techniques.

2.7 Discussion

An important problem we could not address in this chapter is missing information, including but not limited to false positive alarm volume, response time, and raw data.

False positive alarm volume is an important indicator of manpower needed for using the EDR system [44], [45]. Low false positive volume means most alarms are true positive so that the system administrator can focus on mitigation. On the other hand, a high false alarm volume means many of the alarms are false positive alarms. Hence, the system administrator must discover true positive alarms before mitigating attacks, often leading to a needle-in-a-haystack problem. Response time indicates the time elapsed between compromises and alarm generation, which measures the real-time capabilities of the EDR systems. Low response time means the system can detect threats fast so that the system administrator can keep the loss to a minimum. Although the delayed modifier gives some information about the delayed alarm, it does not provide quantitative data on how long the delay is. Moreover, MITRE only provides the detection results from the EDR systems, not the raw data such as system logs and network events. Missing the raw data prevents new EDR systems from using the same dataset to compare performance with existing ones and limits the information available to researchers when they analyze threats with poor detection coverage. Missing the infor-

mation described above hinders the analysis of EDR systems from many meaningful perspectives. We hope MITRE could include them in the future release of evaluations.

Another concern is the prospective compatibility of our interpretation framework within the context of the MITRE evaluation. While MITRE implemented notable modifications to its evaluation framework during the initial three rounds, the framework employed in the latest three rounds has demonstrated sustained consistency. Our interpretation comprehensively encompasses all elements that have maintained this consistency throughout these rounds. Therefore, we are confident that our interpretation framework will remain pertinent and enduring in the foreseeable future.

2.8 Conclusion

By leveraging MITRE's evaluation efforts and introducing our analysis method, we offer valuable insights into the current capabilities of industrial EDR systems to bridge the gap between MITRE's raw evaluation results and comprehensive interpretations. This research aids researchers, practitioners, and vendors in understanding the strengths, limitations, and areas for improvement of EDR systems, ultimately enhancing enterprise security.

2.A Appendix

2.A.1 Ethics

This study does not raise any ethical issues. All the datasets we used are publicly available and anonymized.

2.A.2 Additional graphs

Fig. 2.5 is another causal relationship attack graph we constructed. Fig. 2.6 shows the distribution of visibility and analytic scores of all EDR systems. Fig. 2.7 shows the same distribution of all techniques.

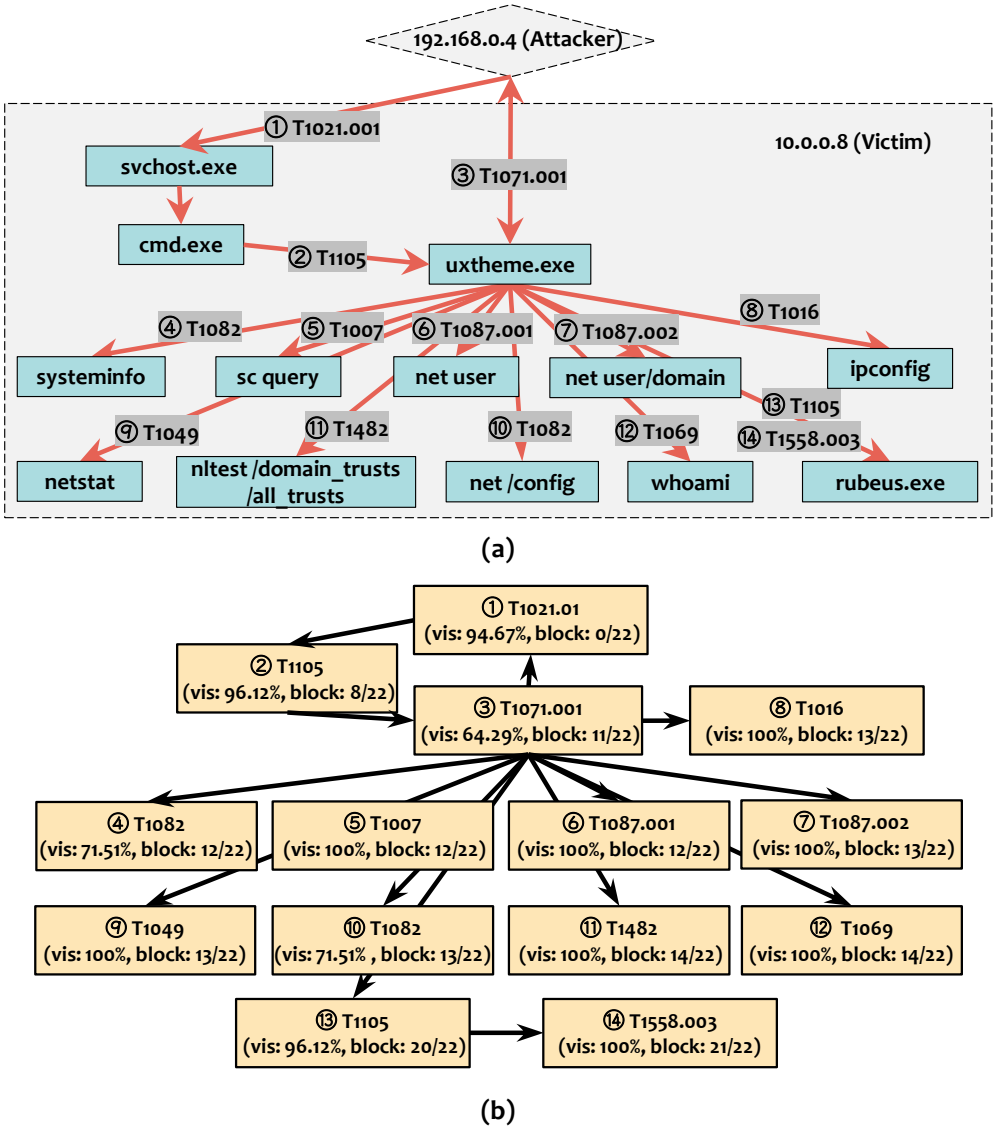


Figure 2.5: The actual attack graph and the causal relationship attack graph for scenario 2 in Wizard Spider+Sandworm (2022) evaluation.

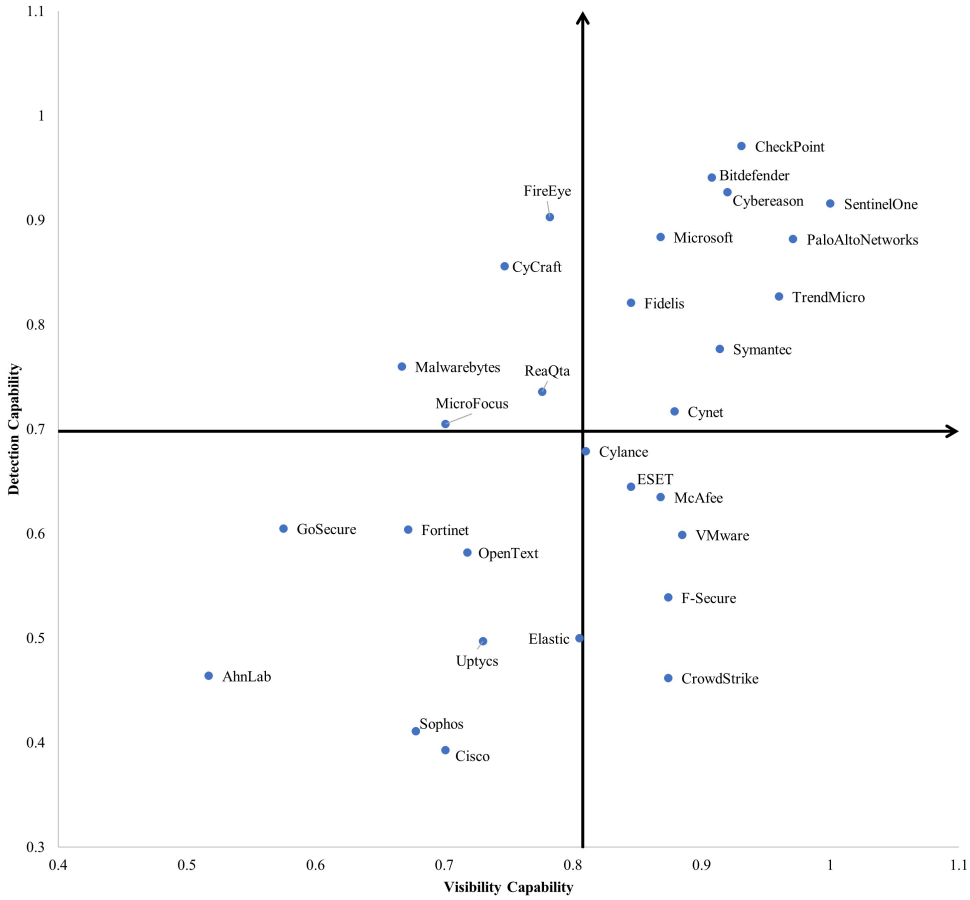


Figure 2.6: Analytics vs. Visibility Quadrant for EDR Systems

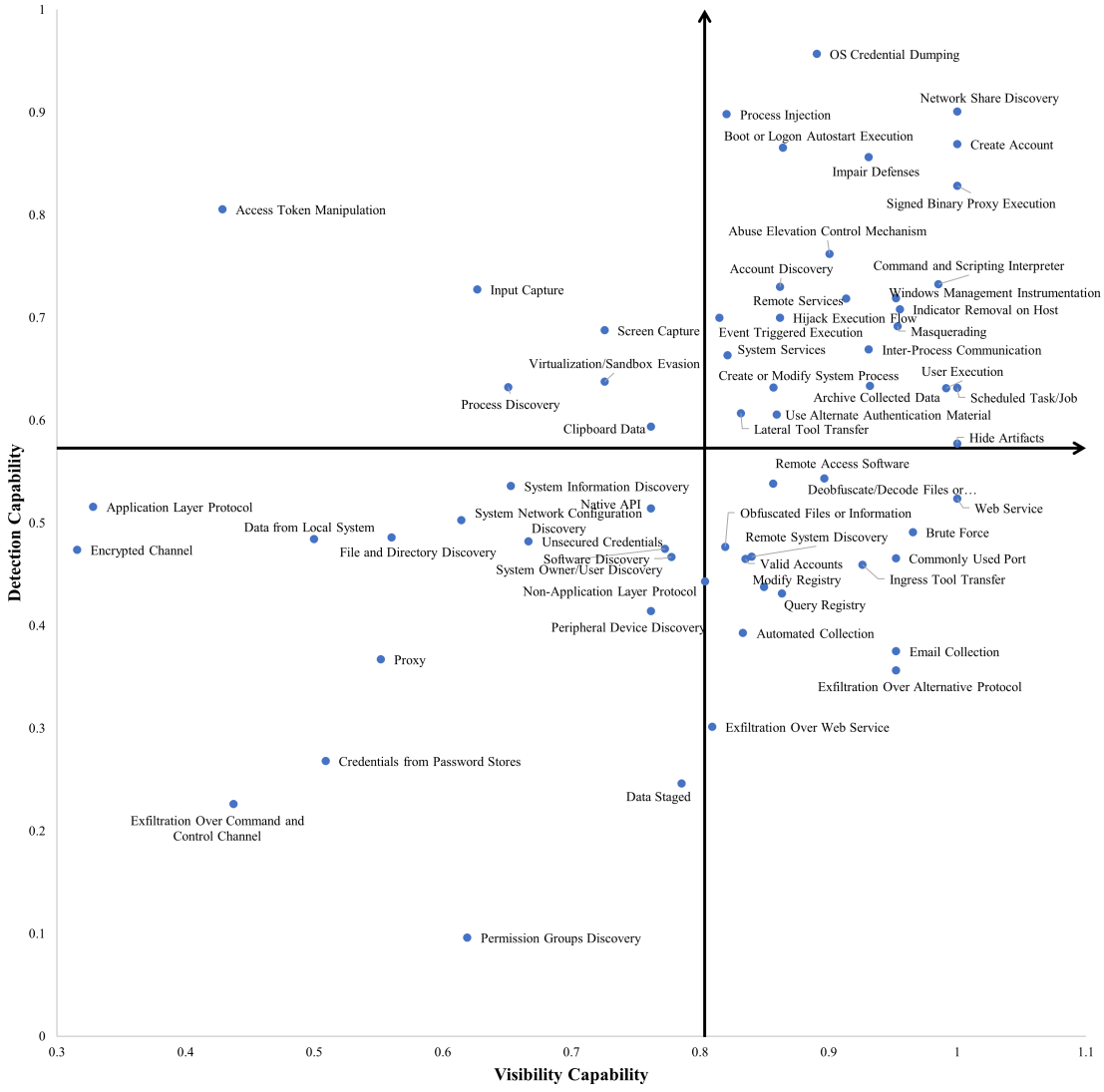


Figure 2.7: Analytics vs. Visibility Quadrant for Techniques

CHAPTER 3
PENTESTAGENT: INCORPORATING LLM AGENTS TO AUTOMATED PENETRATION TESTING

3.1 Introduction

Penetration testing is a widely adopted technique for proactively identifying security vulnerabilities. This process involves gathering information about the target system (reconnaissance), identifying possible entry points, attempting to exploit the system, and reporting the findings. [46] Traditionally, penetration testing has been a complex manual process requiring highly skilled security specialists with extensive experience. Testers typically write their own exploits, master public domain tools, and perform numerous tedious and time-consuming tasks. [47] According to Rapid7's Under the Hoodie report, penetration testing takes an average of 80 hours, with significant outliers taking several hundred hours. [48] Consequently, manual penetration testing often necessitates large, diverse teams, which most organizations cannot afford.

Although automated penetration testing has been a concept for over a decade, a significant gap remains between the proposed methods and their real-world application. Early works [49]–[51] primarily modeled attack planning as an attack graph problem [52] in a deterministic and fully observable world. However, such an approach imposes limitations: it assumes complete observability from the defenders' standpoint and lacks the flexibility and adaptability required for dynamic environments. Later efforts [53]–[60] addressed these shortcomings by introducing uncertainty into planning methodologies, treating attack planning as a Markov Decision Process (MDP), which model the world as states and actions as transitions between states, with a reward function encoding

the "reward" for moving from one state to another. As extensions to MDP-based approaches, the subsequent works employ partially observable Markov decision process (POMDP) [56], [57] and reinforcement learning algorithms [59], [60] to account for further uncertainty in the environment and action outcomes. These advancements better align with real-world conditions where attackers possess limited knowledge of the target systems. Nevertheless, these probabilistic models focus on establishing a theoretical model for automated pentesting planning and lack the implementation aspect.

Large language models (LLMs) are rapidly evolving, showcasing impressive capabilities in a wide range of tasks, including text summarizing, data analysis, and question-answering. The powerful LLMs have gained significant attention in security applications, leading to a shift towards LLM-based security solutions that offer enhanced intelligence and automation capabilities compared to existing methods, making it possible to address the implementation gap in automated penetration testing.

Recent attempts to utilize LLMs for automating penetration testing [61]–[63] have shown some promising initial results. However, two crucial gaps still need to be addressed for practical use:

1) Limited pentesting knowledge: These methods heavily rely on pre-trained language models for generating actionable items. However, the training datasets for these models often lack comprehensive coverage of penetration testing techniques. This results in a limited state space and an outdated action space, reducing the effectiveness and relevance of the generated actions.

2) Insufficient Automation: Existing approaches lack the automated capabilities, including validating and debugging the suggested procedures and dynamically acquiring and applying new pentesting techniques.

To overcome these challenges, we propose a novel LLM-based automated penetration testing framework PENTESTAGENT. Our framework aims to enhance penetration testing knowledge by

Table 3.1: Comparison of LLM-based pentesting systems

System	State&Action Space	Online Search Augmentation	Validation& Debugging Capability
PENTESTAGENT	Large	Auto	Auto
AUTOATTACKER [63]	Unknown[1]	Manual	Manual
PENTESTGPT [61]	Unknown[1]	Manual	Manual
Happe et al. [62]	Small	No	No

[1] AUTOATTACKER and PENTESTGPT solely rely on LLMs to provide reconnaissance and attack techniques, which can be limited and outdated.

continuously integrating new techniques and updating the framework's knowledge base with the assistance of LLMs. Additionally, PENTESTAGENT establishes a robust automated penetration testing pipeline utilizing LLM techniques, incorporating validation and debugging mechanisms to ensure the effectiveness and relevance of generated actions in specific target environments. By bridging these gaps, we aim to significantly improve the practical applicability and reliability of automated penetration testing frameworks.

PENTESTAGENT employs a multi-agent design where each agent is responsible for a specific task in the penetration testing process. This flexible architecture enables the customization of toolsets across different tasks, enhancing the system's adaptability. In addition to LLM agents, PENTESTAGENT integrates Retrieval Augmented Generation (RAG) [64] to leverage supplementary data during response synthesis and manage the context efficiently. This combination enhances the quality of the generated outputs and further reduces the need for manual intervention.

Table 3.1 provides a comparative overview of existing automated pentesting systems, illustrating that while frameworks like AutoAttacker and PENTESTGPT rely on manual or limited approaches, PENTESTAGENT offers a fully automated solution with a larger state and action space and advanced online search augmentation and debugging capabilities.

Moreover, our contributions extend beyond the framework itself. We introduce a comprehen-

sive penetration testing benchmark based on VulHub's collection of vulnerable Docker environments and Capture The Flag (CTF) challenges on HackTheBox. This benchmark spans various difficulty levels and encompasses a wide range of vulnerabilities, addressing a critical gap in current research by providing a practical and accessible evaluation framework.

To sum up, we make the following contributions:

- We design PENTESTAGENT, a LLM-based automated pentesting system that operates with minimal human intervention. PENTESTAGENT integrates multi-agent design and Retrieval Augmented Generation techniques to enhance penetration testing knowledge and automate various tasks.

- We design a comprehensive penetration testing benchmark based on the leading open-source collection of pre-built vulnerable Docker environments VulHub. This benchmark spans various levels of difficulty and encompasses a wide range of common weaknesses and vulnerabilities, providing a comprehensive and practical framework for evaluating penetration testing tools.

- We design experiments and metrics to evaluate PENTESTAGENT on our benchmark. The results demonstrate PENTESTAGENT's superior performance in automatically completing the entire penetration testing process, as well as in individual penetration tasks.

We make our benchmark datasets and framework publicly available to facilitate further research in automated penetration testing.[1]

[1] `https://github.com/nbshenxm/pentest-agent`

3.2 Background and Related Work

3.2.1 Penetration Testing

Penetration testing (pentesting) is a structured, multi-stage process designed to identify security vulnerabilities in systems. According to the Penetration Testing Execution Standard (PTES) [65], pentesting consists of three main stages: intelligence gathering, vulnerability analysis, and exploitation. Penetration testing is broadly divided into external and internal assessments [48]. External assessments focus on assets exposed on the Internet, such as web applications, online services, and external networks, using techniques like social engineering, red teaming, and, in particular, web penetration testing. In contrast, internal assessments target the organization's internal network, source code, or physical devices, typically involving code reviews and internal network compromises.

While several recent works have enhanced internal assessments using LLM-based frameworks (e.g., ChatAFL [66], FuzzGPT [67], LLift [68], and LATTE [69]), external assessments, especially web penetration testing, remain underexplored. According to Rapid7's latest report [70], external network compromise constitutes over 80% of penetration testing tasks. This chapter addresses this gap by demonstrating how PENTESTAGENT automates web penetration testing, thereby enhancing the overall applicability and efficiency of automated pentesting in real-world scenarios.

Existing Tools. While automated tools exist for individual tasks within these stages, integrating them into a seamless and effective workflow remains a significant challenge. Existing tools specialize in specific penetration testing tasks. For instance, Nmap[71] is widely used for intelligence gathering, allowing testers to analyze network configurations through direct interaction with targets. Nessus[72] and OpenVAS[73] focus on vulnerability analysis, scanning systems for known weaknesses using extensive vulnerability databases. Metasploit[74] is commonly used for

exploitation, providing a range of exploits and payloads to execute attacks on identified vulnerabilities. Although these tools are effective alone, their effective use requires expert knowledge, manual decision making, and significant effort to coordinate workflow.

AI-Driven Penetration Testing. Recent advancements in artificial intelligence have led to the development of more sophisticated penetration testing frameworks based on machine learning and Markov Decision Process (MDP) algorithms [58]–[60]. For example, Chen et al. [60] designed a reinforcement learning-based framework for automated attack planning. This framework incorporates expert knowledge into state-action pairs and employs a reward function to train the system to execute actions with the highest success rate. Although these frameworks can generate reasonable attack plans, they lack the dynamic implementation aspects of penetration testing. They are unable to react to potential failures and adjust the plan in real time.

LLM-Based Penetration Testing. The rise of LLM-based applications has further advanced the automation of penetration testing tasks such as text analysis, task planning, code modification, and execution debugging. However, the existing LLM-based penetration testing frameworks still lack comprehensive coverage of the stages and automation for practical use. AUTOATTACKER [63] focuses on constructing post-breach attacks, neglecting the pre-compromise stages. PENTESTGPT [61], while implicitly considering multiple stages through its "pentesting task tree," still relies on human decision to proceed with a certain branch of tasks, leading to inefficiency and ineffectiveness. For example, PENTESTGPT may overly focus on one task while neglecting others, resulting in an unbalanced approach. Moreover, PENTESTGPT and AUTOATTACKER depend on the LLM's pre-trained knowledge and human analysis to gather additional information about the target, discover and validate vulnerabilities, and select the next steps from the task tree. These tasks still demand considerable manual effort.

Our objective is to develop a comprehensive and automated penetration testing framework that

integrates all stages into a coherent and effective workflow. By leveraging the advanced capabilities of LLMs, we aim to significantly enhance the level of automation in penetration testing, reducing dependency on human expertise and effort while ensuring seamless integration across all stages of the penetration testing process.

3.2.2 Challenges of Applying LLM to Pentesting

Although LLM-based systems have demonstrated excellent capabilities in various tasks, we identify the following challenges in using LLM in penetration testing tasks.

C1. Limited Pentesting Knowledge. The LLM has basic knowledge about vulnerabilities and penetration testing to help get the penetration testing started. However, it requires the user to manually search for additional information, such as actual CVE numbers, analyze related information about the CVE like the vulnerability type and relevant exploits, manually set up the exploitation tools, search for the appropriate exploit to run, and configure the exploit. All these additional efforts require penetration testing expertise and significant human effort. For example, here is an example conversation with GPT-4 on how to conduct penetration testing on ActiveMQ 5.17.3.

Limited Pentesting Knowledge Example

User: What may be the vulnerabilities of ActiveMQ 5.17.3? And how can I test if the vulnerabilities exist?
GPT-4:

1. Keep Software Updated ...
2. Check CVEs ...
3. Security Scanning Tools ...
4. Configuration Review ...
5. Penetration Testing ...
6. Log Analysis ...

User: Could you give detailed steps for pentesting?
GPT-4:

1. Identify Known CVEs ...
2. Set Up Metasploit ...
3. Search for ActiveMQ Modules ...
4. Configure and Run Exploit ...
5. Manual Testing ...

C2. Short-term Memory. The limitation of models' context windows, leading to the short-term memory problem, becomes particularly challenging during long-lasting tasks such as penetration testing, which requires continuous memory across a prolonged time period. For instance, in vulnerability analysis, information gathered during intelligence gathering is crucial for identifying vulnerabilities and searching for corresponding exploits. Similarly, in the exploitation stage, information from the intelligence gathering stage aids in selecting and configuring appropriate exploits. The short-term memory limitations can lead to several issues throughout the penetration testing process.

1) Repetition of Tasks: Due to the restricted context window, the model may forget previously gathered information or actions taken, leading to redundant tasks being performed. For example,

LLM may repeat the information collection process that was already performed earlier.

Repetition of Tasks Example

Intelligence Gathering

LLM: Use Nmap to perform a comprehensive scan of all ports on the target host to identify open ports and services.
User: {Nmap scan results}

Vulnerability Analysis

LLM: Use Nmap to perform a comprehensive scan of all ports on the target host to identify open ports and services.
User: {Nmap scan results}

2) Loss of Context: As the model's context shifts with each interaction or stage transition, it may lose the contextual understanding necessary for making informed decisions or executing sequential tasks effectively. This can result in suboptimal exploitation attempts or misalignment with the overall penetration testing objectives. For example, LLM may fail to provide detailed instructions on how to execute an exploit due to context loss.

Loss of Context Example

Intelligence Gathering

{Information collection steps}...
LLM: The target OS is Linux and the target IP is 192.168.238.129.

Exploitation

User: How do I execute this exploit?
LLM: The target OS and IP are needed to configure the exploit. For investigation of the unknown OS and IP, do the following: ...

C3. Workflow integration. In the context of penetration testing, which involves a multi-stage pipeline of interconnected tasks, integrating an LLM introduces several challenges related to output quality control and stateful working memory management.

1) Output Quality Control: Ensuring that the LLM's output is formatted in a way that downstream modules can parse easily is crucial for the smooth operation of the entire penetration testing pipeline. This requires the LLM to generate output in a structured format that adheres to predefined standards or protocols, making it easier for subsequent modules to process and utilize the information effectively. Additionally, maintaining high content quality is essential. Before passing its output to downstream modules, the LLM should conduct validation checks to ensure the accuracy, completeness, and relevance of the generated information. LLMs may suffer from the hallucination problem, producing irrelevant or incorrect answers. Implementing robust quality control is necessary to mitigate the risk of propagating errors or misleading data through the pipeline, thereby reducing the likelihood of a single point failure disrupting the entire testing process.

2) Stateful Working Memory Management: Each stage of penetration testing often requires different sets of stateful working memory, encompassing information such as discovered vulnerabilities, selected exploits, target environment details, and ongoing session contexts. The challenge lies in enabling smooth transitions of this working memory between tasks and sessions. If the LLM cannot retain and switch between continuous stateful memory throughout the penetration testing process, it can disrupt the flow and coherence of the testing sequence. For example, if the LLM fails to retain the progress made in exploit execution after obtaining necessary information from the target environment details working memory to proceed, it may lead to restarting the exploit execution from the beginning. This redundancy can delay progress and impact the overall thoroughness and effectiveness of the testing. However, current LLMs do not inherently support such working memory management within and between sessions, posing a significant challenge in

achieving seamless integration across the penetration testing pipeline.

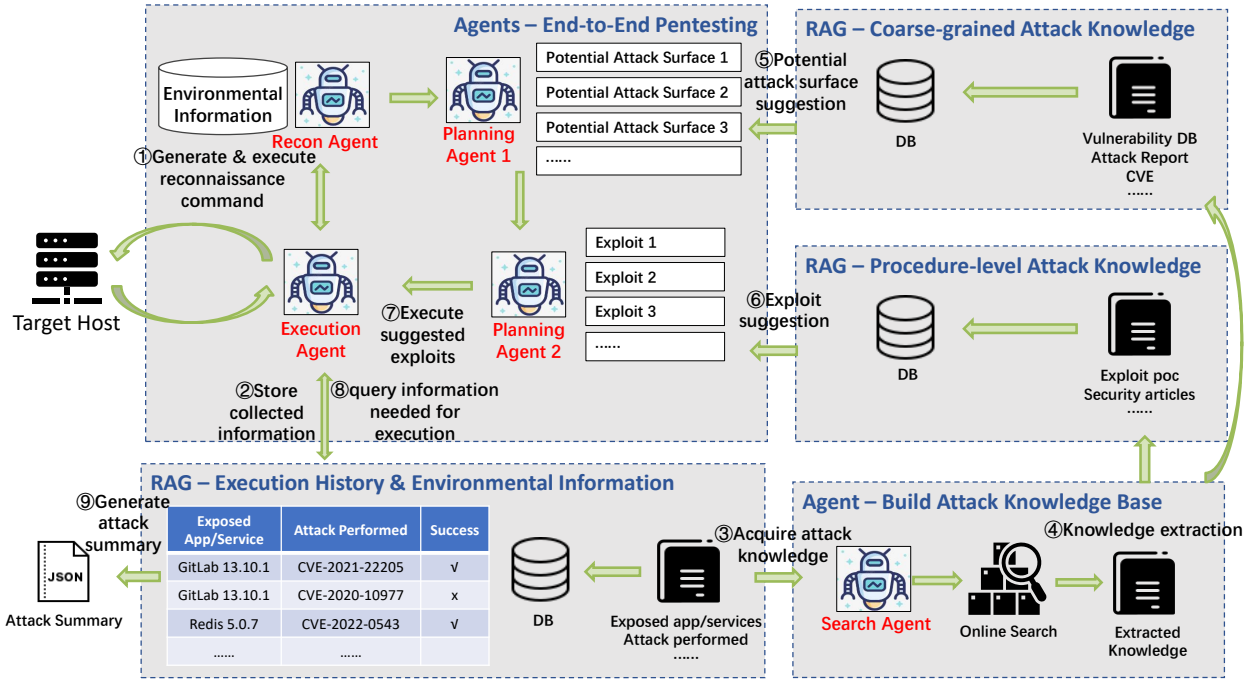


Figure 3.1: An overview of the components in PENTESTAGENT

3.2.3 LLM Techniques for Overcoming Challenges

The rapid advancement of LLM studies has introduced a new level of intelligence and automation capabilities, significantly enhancing penetration testing performance. Various LLM techniques can be applied to different stages of pentesting to improve efficiency and effectiveness, addressing the challenges mentioned in Section 3.2.2.

LLM agents, which are LLMs equipped with additional tools, extend the functionalities of traditional models. These agents can be beneficial in all stages of pentesting by performing tasks that traditionally required human intervention, such as text analysis and code debugging. With the right tools, an LLM agent can search for and learn penetration testing knowledge online, thus addressing the challenge of limited pentesting knowledge (**C1**). To fully leverage an LLM agent's

capabilities, it is essential to provide an appropriate system message that defines the agent's basic profile, including its capabilities, limitations, output format, and additional specifications [75].

Retrieval-augmented generation (RAG) enhances LLMs by allowing them to utilize external data for generating responses. This technique involves three main stages: indexing, retrieval, and response synthesis. Initially, the dataset is indexed for efficient retrieval. Upon receiving a query, RAG retrieves relevant information from the indexed dataset and combines it with the original query before sending it to the LLM for response synthesis. RAG effectively addresses the challenges of short-term memory (**C2**) and stateful working memory management (**C3.2**) by enabling users to maintain long-term memories that can be dynamically queried and stored.

The chain-of-thought (CoT) technique significantly improves the ability of large language models to perform complex reasoning [76]. By guiding the LLM to follow a logical sequence of steps, this method enhances the model's problem-solving capabilities.

Role-playing [77] asks the LLM to impersonate an imaginary character, allowing LLM to operate with clear objectives and boundaries, thereby enhancing their efficiency and effectiveness.

Self-reflection techniques, where the LLM summarizes its past mistakes into long-term memory to avoid similar errors in subsequent communications, have proven useful for learning complex tasks over a handful of trials [78].

Structured output techniques can save time spent on iterative prompt testing and ad-hoc parsing, reducing overall LLM inference costs and latency, as well as developers' effort. Additionally, structured outputs ensure smooth integration with downstream processes and workflows [79].

Together, these techniques significantly improve the quality of LLM output, effectively addressing the output quality control challenge (**C3.1**).

3.3 System Design

3.3.1 System Overview

As shown in Fig. 3.1, PENTESTAGENT comprises four major components: the **reconnaissance agent**, the **search agent**, the **planning agent**, and the **execution agent**. These agents collaborate to perform the three main stages of penetration testing.

Intelligence Gathering: ① Upon receiving user input specifying the target, the reconnaissance agent initiates the penetration testing process by gathering environmental information about the target host. The reconnaissance agent generates and executes reconnaissance commands, aiming to collect comprehensive environmental data from the target host. ② The reconnaissance agent then analyzes the execution results and compiles a summary of the target environment, which is stored in a designated environmental information database.

Vulnerability Analysis: Next, the search and planning agents work together to perform the vulnerability analysis. ③ The search agent queries the environmental information database to retrieve a list of services and applications exposed on the target host. ④ Guided by these services and applications, the search agent searches for potential attack surfaces and procedures and saves them in separate databases. ⑤ The planning agent first leverages the RAG techniques to find a list of potential attack surfaces. ⑥ Subsequently, the planning agent uses these identified attack surfaces to determine suitable exploits for the target environment.

Exploitation: ⑦ Finally, the execution agent attempts to execute these attack plans on the target host. ⑧ The execution agent communicates with the environmental information database to obtain the necessary information for executing the exploits. It also debugs any execution errors by modifying the code or executing additional commands to gather more information. ⑨ All execution history is stored in a database and can be used to generate a comprehensive penetration testing

report.

This structured and automated framework aims to streamline the penetration testing process, enhancing efficiency and reducing the manual effort required.

3.3.2 Reconnaissance Agent

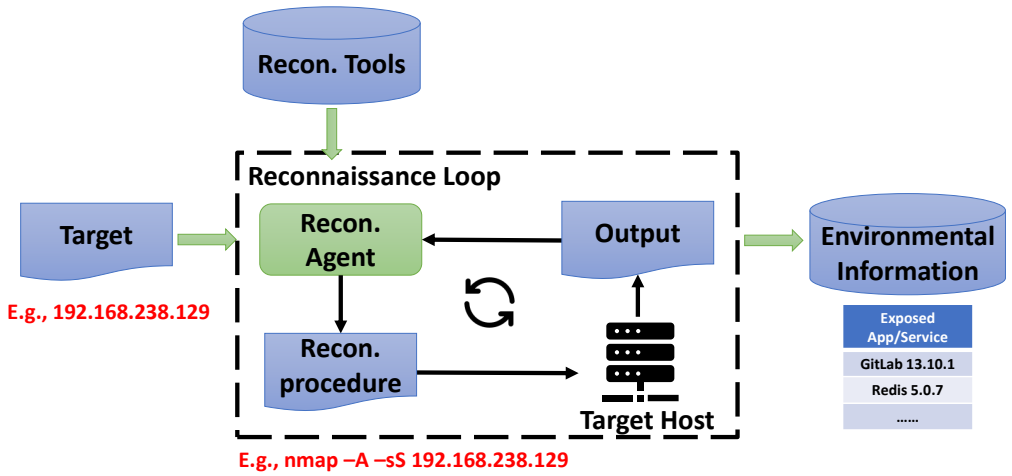


Figure 3.2: Reconnaissance agent workflow

The reconnaissance agent takes a specified target as input and interacts with it to collect detailed information, ultimately generating a summary of the environmental information as the output. As illustrated in Fig. 3.2, the process begins when a target is provided to the reconnaissance agent. The agent operates in a self-iterating loop, generating reconnaissance commands to gather information from the target and analyzing the results of these commands until the best efforts have been made. Once the reconnaissance loop concludes, the agent summarizes its findings and stores them in a database.

The reconnaissance agent adheres to a general workflow defined with expert knowledge to perform the reconnaissance task. It determines specific procedures or tools to use with the help of external knowledge supported by the RAG framework. To achieve our desired workflow, we

carefully design the system messages and prompts for the reconnaissance agent, implementing the following techniques to overcome the challenges mentioned in Section 3.2.2.

Reconnaissance System Message (Simplified)

Role-play

You're an excellent cybersecurity penetration tester assistant. Guide the tester ...

Chain-of-Thought

Use Nmap to identify exposed ports, then use relevant tools in Nmap to analyze these ports on the target host ...

RAG

You should use your query tool to learn about available reconnaissance tools ...

Structured Output

You should always respond in valid JSON format with the following fields: {FORMAT SPEC.} ...

Role-playing has proven effective in bypassing the safety policies enforced by the LLM [80]. Thus, we ask the reconnaissance agent to act as a penetration tester assistant to validate its reconnaissance behaviors.

We use Chain-of-Thought (CoT) to break down complex tasks into several sub-tasks and construct an effective reconnaissance workflow to reduce hallucination. Since the reconnaissance workflow involves a self-iterating loop, it is important to specify a stop condition to avoid the agent getting into an infinite loop. Using CoT effectively enforces the stop condition by specifying the tasks to complete before stopping.

Retrieval-Augmented Generation (RAG) allows the reconnaissance agent to retrieve relevant information from a database containing documentation of various reconnaissance tools, enabling it to use up-to-date tools for effective information collection. For example, it can use web application

fingerprinting tools with open-source fingerprinting databases like ObserverWard [81] to aid in reconnaissance. Furthermore, RAG allows the reconnaissance agent to store collected environmental information in a database for later use, addressing the short-term memory issue.

The reconnaissance agent analyzes previous execution results and generates the next command to execute in each communication. To enforce adherence to the penetration testing pipeline and ensure a smooth transition to subsequent steps, we use structured output, asking the reconnaissance agent to respond using a specified format.

After the reconnaissance agent determines that it should stop the reconnaissance loop, it summarizes the reconnaissance results and stores them in a database to make the short-term reconnaissance memory persistent.

3.3.3 Search Agent

The search agent takes target services and applications as input and stores relevant attack knowledge into databases as output. As illustrated in Fig. 3.3, the search agent performs two rounds of hierarchical online search for relevant information. In the first round, it searches and analyzes the results to extract potential attack surfaces relevant to the target. In the subsequent round, it uses the identified potential attack surfaces as a guide to search and analyze procedure-level attack knowledge. The potential attack surfaces and procedure-level attack knowledge are stored in two separate databases for future use.

The online search module is customizable and extensible. We have implemented several search functions, including general searches on Google, vulnerability-specific searches on databases like Snyk [82] and AVD [83], and searches in exploit code repositories such as GitHub and ExploitDB. In our hierarchical search workflow, we use Google and vulnerability database searches to identify potential attack surfaces in the first round and then employ Google and code repository searches to

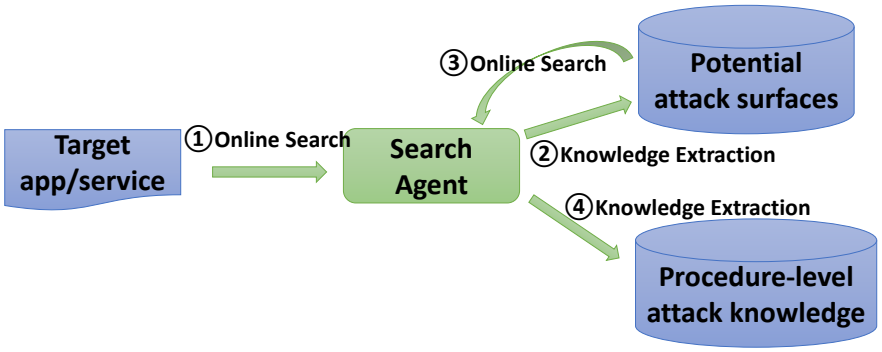


Figure 3.3: Search agent workflow

find exploit implementation details in the second round.

After each round of online searches, the search agent analyzes the results. However, indexing and storing information from raw search results is inefficient. Therefore, we leverage RAG-based question-answering to extract key information from the raw search results and use the extracted knowledge to build a more relevant and concise database. As elucidated in Fig. 3.4, given the analysis prompt, the RAG framework will first retrieve relevant segments of information from the search results. Then, it sends the analysis prompt with the retrieved information as context to the LLM as the question, and the LLM will analyze the information in the context to help answer the queries in the analysis prompt and generate a comprehensive summary for the search results containing the key information we are looking for. Finally, the summaries of individual documents are gathered to build a potential attack surface or exploit database in Fig. 3.3.

For the first round of searching for potential attack surfaces, we use the following prompt to extract knowledge from individual search results. Specifically, we ask for relevancy and key information about vulnerabilities, such as CVE numbers, as well as other keywords or URLs that can lead to more detailed information. We also ask the search agent to output the analysis results in a structured format for subsequent processing.

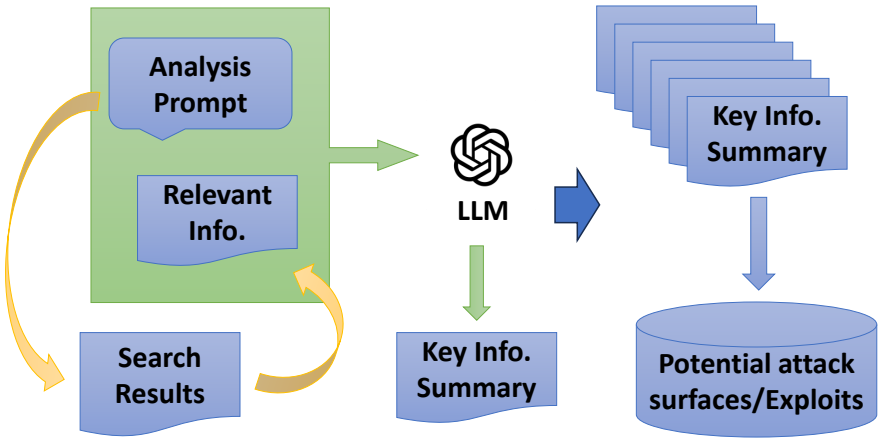


Figure 3.4: RAG workflow for search result summarization. The yellow arrows denote the retrieval process, and the green arrows denote the generation process.

Potential Attack Surface Analysis Prompt (Simplified)

RAG & CoT

Generate a concise summary of the document to answer the following questions:
1) Does this document describe vulnerabilities targeting a particular service or app; if so, what is the relevant service/app version?
2) Provide information that can be used to search for the exploit of the vulnerabilities.

Structured Output

You should always respond in valid JSON format with the following fields: {FORMAT SPEC.} ...
For example, the response looks like this: {OUTPUT FORMAT EXAMPLE}

Similarly, for the second round of searching for procedure-level exploit details, the search agent analyzes individual search results using RAG and CoT. First, it checks whether the repository contains a relevant exploit. Then, it extracts key information such as applicable service or application versions and prerequisites for running the exploit. While the first round of analysis mainly focuses on the LLM's text summarization capability, the second round relies on the LLM's code analysis capability to determine whether the code functions as an exploit and the dependencies required to

execute it.

After the penetration testing knowledge is extracted by the search agent, it is stored in a hierarchical tree structure as shown in Fig. 3.5. The hierarchical tree-structured penetration testing knowledge base allows efficient searching and systematic management of penetration testing knowledge.

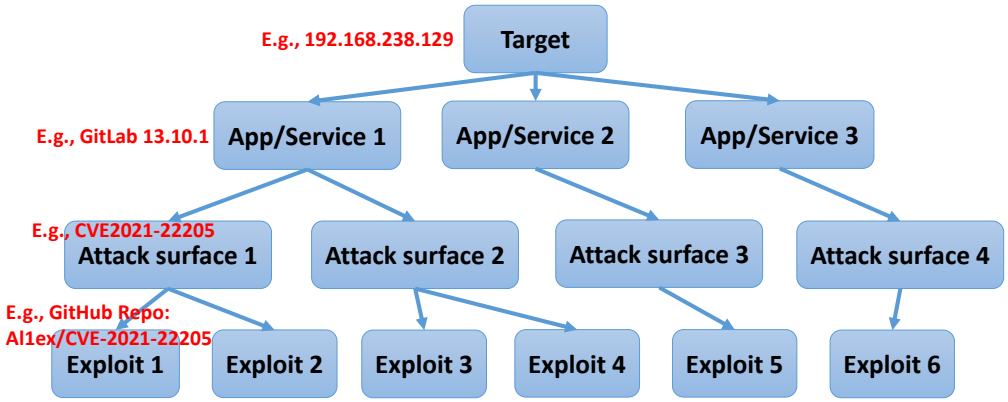


Figure 3.5: Hierarchical pentesting knowledge database

3.3.4 Planning Agent

The planning agent takes the detected services and applications from the reconnaissance agent as input and generates an exploitation plan as output. As shown in Fig. 3.1, the planning agent leverages RAG and the pentesting knowledge base (Fig. 3.5) to first generate a list of potential attack surfaces relevant to services and applications. Then, the planning agent follows a similar process to generate a list of exploits.

The planning agent uses the service or application as a key to find the relevant database for potential attack surfaces and retrieves these from the database according to the version of the service or application and the types of vulnerabilities. The planning agent makes suggestions for attack surfaces based on the application version and categorizes attack surfaces by vulnerability types.

The planning agent then uses the attack surface to find the relevant database for exploits and retrieves exploit details from the database according to the service or application version and exploit effects (e.g., remote code execution, authentication bypass). The planning agent then makes suggestions for exploits based on the application version and categorizes the exploits by exploit effects.

3.3.5 Execution Agent

The execution agent takes the details of the exploit as input and attempts to execute the exploit on the target automatically, ultimately generating an exploitation summary as output. The execution agent follows the order suggested by the planning agent. As illustrated in Fig. 3.6, each exploit execution can be divided into two stages: the preparation stage and the exploitation stage.

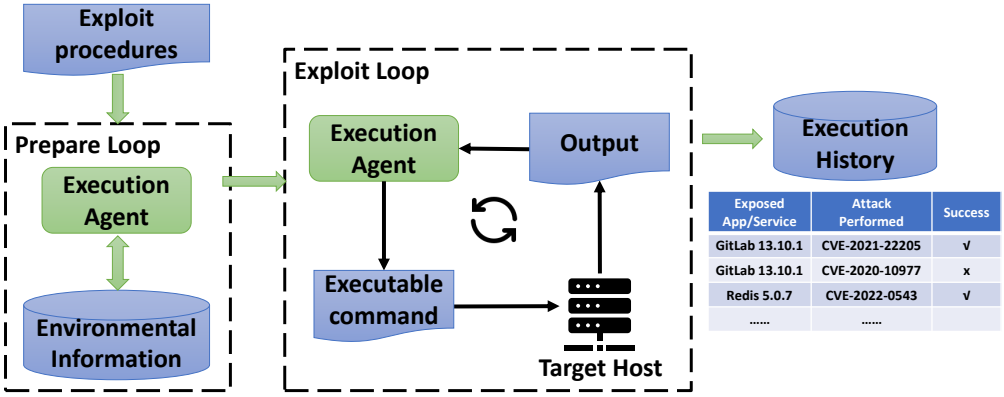


Figure 3.6: Execution agent workflow

During the preparation stage, the execution agent analyzes the details of the exploit to determine the requirements for successful execution, such as the parameters needed. It then queries the database containing environmental information to obtain the necessary information. The following system message guides the execution agent in fulfilling its duty. To bypass the safety mechanisms that may prevent the process while the execution agent tries to execute the exploit, we use role-playing to make it act as a cybersecurity penetration tester.

To effectively find the information needed to execute the exploit, we employ the CoT technique to guide the execution agent to first identify all the parameters and then determine the information needed for each parameter. During this analysis, the execution agent retrieves relevant information from the exploit details using RAG to provide the context. Finally, the execution agent outputs the required information in a structured JSON format, allowing it to proceed to the subsequent step of requesting information from the environmental information database.

Execution Preparation System Message (Simplified)

Role-play

You're an excellent cybersecurity pentesting assistant ...

CoT

Analyze the exploit to answer the following questions:
1) What parameters do you need to fill in to execute this exploit successfully?
2) What information do you need to fill in these parameters?

RAG

You should use your query tool to learn the details about the exploit ...

Structured Output

You should always respond in valid JSON format with the following fields: {FORMAT SPEC.} ...

After obtaining a list of needed information, the execution agent queries the environmental information database to fill in the information. After receiving the response containing the requested information, the execution agent's system message is updated as follows to transition to the exploitation stage.

Execution Exploit System Message (Simplified)

Your next task is to provide step by step guide for executing the exploit and debugging the errors encountered ...

RAG

You should use the tool to learn the code and README of the exploit to figure out how to properly execute it.

Specifications

Avoid commands that require user interactions ...

Self-reflection

When the results indicate an error, you should ...

Structured Output

You should always respond in valid JSON format with the following fields: {FORMAT SPEC.} ...

During the exploitation stage, the execution agent uses RAG to obtain details of the code execution, breaks down the execution plan, and generates a step-by-step execution guide. Similar to the reconnaissance agent, the execution agent engages in iterative loops to execute the exploit.

When errors are encountered during exploit execution, proper error handling is required. To guide the execution agent in debugging errors, we employ the self-reflection technique. The execution agent analyzes and fixes errors based on the code and error message while concurrently documenting the error history for future reference to avoid repeating the error. This iterative process ensures continual refinement and optimization of our automated pentesting system.

3.4 Evaluation

In this section, we present the benchmark established for evaluating automated penetration testing frameworks and discuss the evaluation results. We address the following research questions (RQs) in our evaluation:

RQ1. Effectiveness. What's the success rate of finishing the whole penetration testing process automatically?

RQ2. Completion level. What's the completion level of individual penetration testing stages that can be automatically finished?

RQ3. Efficiency. How much time and API cost are needed for PENTESTAGENT to complete a penetration testing task?

3.4.1 Evaluation Setup

3.4.1 Benchmark Dataset

The benchmark dataset should be easily accessible and include a diverse set of tasks with varying difficulty levels to evaluate the automated penetration testing framework. Accessibility is essential for a good benchmark; otherwise, it prevents the whole community from using it. The tasks in the benchmark should involve exploiting various vulnerabilities targeting different services and applications to mimic real-world penetration testing scenarios. More importantly, the tasks should have appropriate difficulty labels to reflect how well the system under test can handle tasks of different difficulty levels, helping researchers identify the strengths and weaknesses of the system.

Several platforms can serve as the dataset of the benchmark, such as HackTheBox [84], OWASP Benchmark [85], VulnHub [86], and VulHub [87]. OWASP Benchmark and VulnHub contain thousands of target testing environments, covering a wide range of real-world penetration testing

scenarios. However, setting up these environments for testing requires significant human effort. Furthermore, they do not provide a difficulty level reference for their test cases, necessitating manual effort to determine the difficulty level for each test case.

Finally, we chose VulHub and HackTheBox as our benchmark dataset. VulHub provides an open-source collection of over a hundred pre-built vulnerable Docker environments, which has been widely recognized and utilized in penetration testing practices. The container-based platform supports infrastructure as code (IaC), making it easy to set up the testing environments. Besides, Docker containers provide sufficient isolation for penetration testing. Moreover, most vulnerable environments in VulHub are constructed to reproduce a particular Common Vulnerabilities and Exposures (CVE) [88]. Each vulnerable environment is associated with a CVE number, which allows us to use metrics associated with CVE numbers to learn about the properties of each vulnerable environment. Specifically, we learn about the difficulty of vulnerability exploits through the Common Vulnerability Scoring System (CVSS)[89] and learn about how realistic the vulnerable environment is via the Exploit Prediction Scoring System (EPSS)[90]. We elaborate on how we construct the benchmark dataset in Section 3.A.2 in the appendix.

As a result, we compiled a benchmark comprising 67 penetration testing targets, spanning 32 CWE (Common Weakness Enumeration) categories as shown in Fig.3.13 in the appendix. These vulnerabilities cover eight security risks in OWASP Top 10 vulnerability [91]. Within our benchmark, there are 50 targets with easy exploitability difficulty, 11 with medium exploitability difficulty, and 6 with hard exploitability difficulty. In addition, we incorporated 11 Capture The Flag (CTF) challenges from HackTheBox to simulate more challenging and realistic scenarios. These challenges are used in Section 3.4.5 for a practicality study and in Section 3.4.6 for the comparative evaluation with PentestGPT. This diverse and realistic collection of vulnerable environments ensures a comprehensive assessment.

3.4.1 Metric

To answer our research question, we design metrics to evaluate the effectiveness and efficiency of PENTESTAGENT. These metrics are essential for assessing the performance of the automated penetration testing framework.

We measure the effectiveness of PENTESTAGENT by determining whether all three stages of penetration testing are completed successfully and automatically. We define successful completion as follows: given a target IP, PENTESTAGENT can automatically perform a functional exploit on the vulnerable environments. For the HackTheBox targets, our focus is on obtaining initial access to the target host. In this context, a successful exploit is one that grants access to the target system.

Some penetration tests may be partially successful and require human assistance. However, failure in a previous penetration testing stage will affect the subsequent stages. To better understand the effectiveness of each component in PENTESTAGENT, we measure the completion level at the stage level. This involves assessing the penetration testing stages that can be completed, assuming the preceding stages have been successful. The completion criteria for each stage are defined as follows. The information gathering stage is considered complete if the target application is successfully identified by PENTESTAGENT. The vulnerability analysis stage is marked as complete when PENTESTAGENT identifies functional exploits based on the target application. We manually verify whether the discovered exploits are effective. The exploitation stage is completed if PENTESTAGENT can automatically and successfully execute the exploit. This stage-level evaluation provides a granular understanding of PENTESTAGENT 's autonomy and effectiveness in progressing through the penetration testing process with minimal human assistance.

Furthermore, we measure the efficiency of PENTESTAGENT using the time taken and the API cost incurred to complete penetration tests. The time metric evaluates the duration required for PENTESTAGENT to complete an entire penetration test cycle, from initial reconnaissance to exploit

execution. The API cost metric quantifies the computational resources consumed by the framework during the testing process. These metrics provide insights into the system's resource consumption and operational speed, which are critical for practical deployment and scalability.

Table 3.2: Summary of LLM models used in our evaluation. Input/output costs are based on pricing at the time of testing.

Model	Context Window	Knowledge Cutoff	Input Cost	Output Cost
GPT-3.5-turbo-0125	16,385 tokens	Sep 2021	$0.50/1M tokens	$1.50/1M tokens
GPT-4o	128,000 tokens	Oct 2023	$5.00/1M tokens	$15.00/1M tokens
o1-mini	128,000 tokens	Oct 2023	$1.10/1M tokens	$4.40/1M tokens
Llama 3.1-8B-Instruct	128,000 tokens	Dec 2023	Open-source	Open-source

3.4.1 Environment setup

The simulated vulnerable applications are hosted on a virtual machine with 2 CPU cores and 8 GB RAM, running Ubuntu 22.04 LTS. To avoid interference with the testing process, we have disabled all services that require listening on ports, such as SSH. The attacker machine is also hosted on a virtual machine with 16 CPU cores and 16 GB RAM, running Kali Linux 2024.1. The attacker machine includes all the pre-installed tools available in Kali Linux, with no additional tools installed. The victim machine and the attacker machine maintain network connectivity via NAT. The vulnerable containers on the victim machine are created with the network parameter set to the victim machine's IP, allowing the attacker machine to directly access the vulnerable environments hosted in the victim machine's containers. This setup ensures the attacker can simulate real-world network conditions when attempting to exploit the vulnerabilities.

We evaluate our framework using a mix of commercial and open-source LLMs. Table 4.3 summarizes their key properties.

3.4.2 Effectiveness of the Entire Framework

We investigate the effectiveness of PENTESTAGENT by its success rates in completing the penetration testing process. Fig. 3.7 shows the success rates of exploiting vulnerabilities categorized by difficulty levels and overall performance across different models. The GPT-4 model demonstrated a 74.2% overall success rate in completing automated penetration testing tasks, outperforming the GPT-3.5 model, which achieved a 60.6% success rate. Both models consistently achieved success rates above 60%, affirming the effectiveness of PENTESTAGENT in establishing an automated penetration testing pipeline.

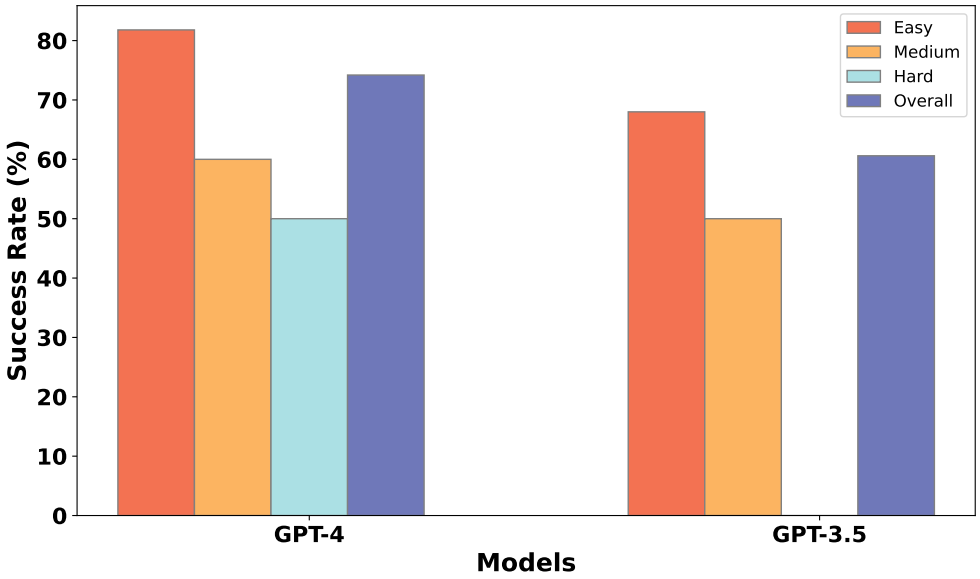


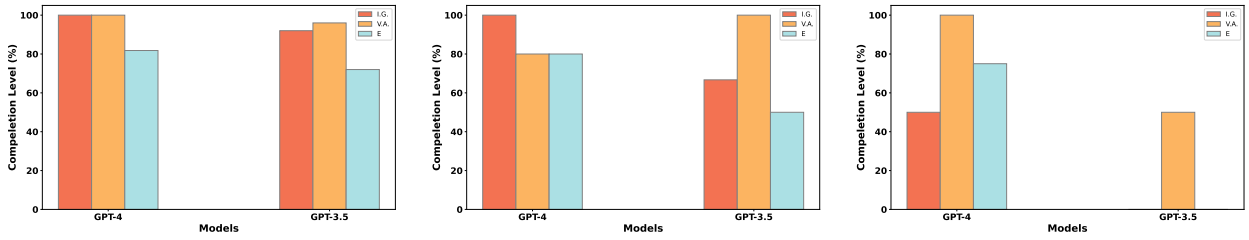
Figure 3.7: Success rate on penetration testing tasks

While the GPT-4 model showed a higher overall success rate compared to GPT-3.5, the difference between their performances was not substantial. This suggests that our framework does not rely heavily on LLMs' general knowledge and capabilities alone.

Notably, the GPT-3.5 model struggled particularly with hard penetration testing tasks, achieving no success in the hardest category. This disparity likely stems from the inherent differences in context window size and learned knowledge between the models, impacting their ability to handle

the complex reasoning required for challenging tasks.

3.4.3 Completion level of Penetration Testing Stages



(a) Easy tasks (b) Medium tasks (c) Hard tasks

Figure 3.8: Completion level of penetration testing stages on different difficulty levels of tasks. I.G. denotes the intelligence gathering stage, V.A. denotes the vulnerability analysis stage, and E denotes the exploitation stage.

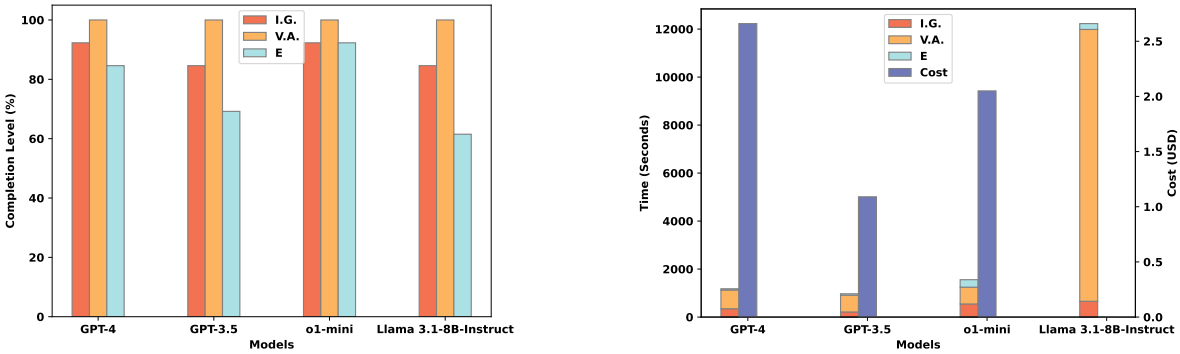
To further evaluate PENTESTAGENT, we analyze its performance across individual penetration testing stages. Fig. 3.8 presents the completion rates for intelligence gathering, vulnerability analysis, and exploitation across different difficulty levels and LLM backbones.

GPT-4 demonstrated robust performance across all stages and difficulty levels, effectively handling a range of penetration testing tasks. In easy tasks, it achieved full completion in both intelligence gathering and vulnerability analysis, with an 81.8% completion rate in exploitation. For medium-difficulty tasks, GPT-4 maintained high completion rates across all stages, with a minor drop in vulnerability analysis. However, in hard tasks, performance declined, particularly in intelligence gathering (50%), suggesting limitations in handling complex reconnaissance tasks that require advanced reasoning and adaptive strategies.

In contrast, GPT-3.5 exhibited more variability across difficulty levels. It performed well in easy tasks, with 92% completion in intelligence gathering and 96% in vulnerability analysis, though its exploitation stage completion rate (72%) was slightly lower. For medium tasks, while

maintaining a 100% completion rate in vulnerability analysis, its performance declined in intelligence gathering (66.7%) and exploitation (50%), indicating challenges in navigating complex reconnaissance and execution scenarios. Notably, in hard tasks, GPT-3.5 struggled significantly, failing to complete both intelligence gathering and exploitation, highlighting its limitations in reasoning and contextual understanding required for advanced penetration testing.

Overall, both models effectively automate significant portions of penetration testing, but GPT-4 consistently outperforms GPT-3.5, particularly in more complex scenarios. These results emphasize the importance of advanced reasoning capabilities and enhanced reconnaissance strategies in achieving higher success rates in automated penetration testing.



(a) Completion level on targets

(b) Average time spent and cost on targets

Figure 3.9: Completion level and overhead of different LLM Backbones.

3.4.4 Ablation Study

To assess how different LLM backbones influence PENTESTAGENT's performance, we performed an ablation study on a subset of VulHub targets. Due to hardware limitations, specifically insufficient GPU resources to efficiently run the Llama 3.1 model on the full dataset, we selected a smaller subset comprising six easy, five medium, and two hard targets. Fig. 3.9a and Fig. 3.9b summarize the completion levels and overhead results, respectively. The results show that while all models perform consistently in vulnerability analysis (with 100% completion), differences emerge

in the other stages. For the intelligence gathering stage, GPT-4 and o1-mini achieve higher completion levels compared to GPT-3.5 and Llama 3.1-8B-Instruct, suggesting that certain models are more effective in integrating context and managing complex reasoning. In the exploitation stage, o1-mini outperforms the others, indicating its strength in executing detailed attack procedures, whereas GPT-3.5 and Llama 3.1-8B-Instruct fall behind.

Overhead measurements further highlight trade-offs between processing time and cost. Although GPT-3.5 is faster in intelligence gathering and is the most cost-effective, its lower exploitation performance suggests a potential compromise in handling complex tasks. In contrast, while o1-mini delivers the best exploitation completion, it incurs longer processing times during vulnerability analysis. The Llama 3.1-8B-Instruct model, despite having no cost, suffers from significant time overhead and lower exploitation performance, which may limit its practical use.

3.4.5 Practicality Study

To evaluate PENTESTAGENT's effectiveness in realistic settings, we deployed it to solve HackTheBox challenges. Unlike standardized benchmarks, these challenges simulate dynamic, real-world penetration testing tasks by presenting diverse vulnerabilities across various CWEs. In this study, we selected 11 HackTheBox machines, nine labeled as easy, one as medium, and one as hard, to provide a broad assessment of the framework's practical utility.

Fig. 3.10 shows the completion level and overhead of PENTESTAGENT on HackTheBox challenges. Table 3.3 in Appendix 3.A.3 further details PENTESTAGENT's performance on these challenges by reporting the number of completed testing stages. While PENTESTAGENT successfully exploited six machines, others like *Pilgrimage* achieved only partial completion, with only the vulnerability analysis stage being successful.

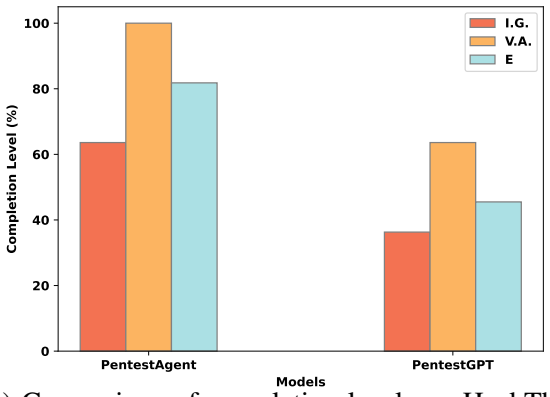
Overall, these results indicate that PENTESTAGENT can address a range of real-world scenar-

ios. However, the variability in stage completion underscores the need for further improvements to achieve more consistent, full-stage automation. We discuss the failed cases in detail in Section 3.4.7.

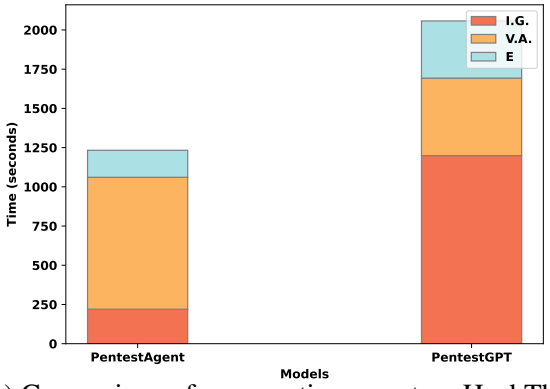
3.4.6 Comparison with PENTESTGPT

We conducted a comparison of the effectiveness and efficiency of PENTESTAGENT against PENTESTGPT. Unlike PENTESTAGENT, PENTESTGPT requires human participation for feedback and decision-making throughout the penetration testing process. Thus, we compare their performance using case studies. We randomly selected ten vulnerabilities from VulHub (five easy, three medium, and two hard) and included 11 HackTheBox challenges (nine easy, one medium, and one hard). Two evaluators with different skill levels conducted the tests: an undergraduate student with limited penetration testing experience evaluated PENTESTGPT on the VulHub targets, while a PhD student with more experience assessed the HackTheBox challenges. Both systems were configured to use the GPT-3.5 model to ensure a fair comparison.

Fig. 3.10 shows the completion level and overhead comparison between PENTESTAGENT and PENTESTGPT on HackTheBox targets. Our results indicate that PENTESTAGENT achieves higher exploitation success and overall efficiency. For instance, PENTESTAGENT completes intelligence gathering in 220 seconds compared to 1199 seconds for PENTESTGPT, and finishes exploitation in 172 seconds versus 364 seconds. Although PENTESTGPT is slightly faster in vulnerability analysis, its slower performance in other stages due to human involvement reduces its overall efficiency. Additional results on VulHub targets and evaluation details can be found in the Appendix 3.A.3. These findings imply that PENTESTAGENT can deliver more consistent and timely penetration testing, highlighting the benefits of minimizing human intervention in real-world security assessments.



(a) Comparison of completion levels on HackTheBox targets



(b) Comparison of average time spent on HackTheBox targets

Figure 3.10: Completion level and overhead comparison on HackTheBox targets.

3.4.7 Failure Analysis

We analyzed failure cases encountered during our evaluation and identified a few representative failure scenarios. As illustrated in Fig. 3.8, most failures occurred during the intelligence gathering and exploitation stages.

In the intelligence gathering stage, PENTESTAGENT occasionally fails to recognize services or applications with the appropriate level of granularity. For instance, our evaluation revealed that PENTESTAGENT struggled to detect components like PHPMailer, PHPUnit, and Ghostscript. These are not standalone applications but rather plugins or components running on web servers. Tools like Nmap can identify the underlying web server frameworks, such as Nginx, but fail to enumerate these components. To address this limitation, PENTESTAGENT allows integration of additional web component fingerprinting tools and specialized libraries to more accurately detect and categorize such web components.

At the exploitation stage, PENTESTAGENT can encounter failures due to several challenges: requiring additional knowledge, needing user interaction, or experiencing LLM hallucinations.

Requiring Additional Knowledge: Certain exploits demand a level of domain-specific knowl-

edge that may exceed the capabilities of an LLM agent. For example, exploiting Samba server 4.6.3 (CVE-2017-7494) assumes the attacker has prior knowledge of credentials (username and password) to establish an SMB connection. Moreover, exploiting JBoss (CVE-2017-12149) requires expertise in using the ”ysoserial” tool to craft payloads for exploiting unsafe Java object deserialization. These limitations can be overcome by integrating a human-in-the-loop design, where human experts can provide the additional knowledge or context required. Thanks to PENTESTAGENT's modular structure and its task-decomposition pipeline, human experts can easily intervene at any point in the testing process to assist with complex tasks.

Requiring User Interaction: Some exploits require user interactions that are typically performed manually, such as file uploads via web user interfaces. For instance, exploiting elFinder (CVE-2021-32682), an open-source file manager for web environments, involves manually creating and uploading an archive file. Similar to the mitigation method in the previous scenario, PENTESTAGENT allows the human user to step in at any penetration testing stage to assist tasks requiring user interaction. Furthermore, the recent advancements in intelligent agents like AutoGPT [92] offer a promising solution by mimicking human actions for complex tasks. By integrating such intelligent agents, PENTESTAGENT could automate these user interactions, significantly enhancing its capabilities in handling tasks traditionally performed by human testers.

LLM Hallucination: Another challenge is LLM hallucination, where the model generates incorrect or misleading information. This issue can be particularly problematic during the exploitation phase, as one hallucination can lead to a cascade of errors in subsequent steps. For example, if the execution agent fails to generate the correct commands or input parameters, it may mistakenly assume the exploit has bugs, leading it down an incorrect debugging path that will never succeed. We employ several strategies to mitigate hallucinations. First, we reduce the randomness of LLM outputs by setting the model's temperature to zero and attempting to execute the exploit

multiple times. We also implement several stop conditions to prevent unintended consequences of hallucination, such as getting stuck in infinite loops or executing unintended actions. These stop conditions include hard-coded limits on the number of execution attempts and prompt-based conditions like "stop when you see the same error again." Additionally, the attack knowledge base usually contains multiple exploits for the same vulnerability, allowing PENTESTAGENT to attempt different approaches until a functional exploit is found.

3.5 Discussion

3.5.1 Comparison with Existing Frameworks

While our primary comparison in this work is with PENTESTGPT, several emerging frameworks address related challenges in automated penetration testing. Notably, AutoAttacker and Enigma offer complementary perspectives that highlight different strengths and limitations relative to PENTESTAGENT.

AutoAttacker [63] focuses exclusively on the post-breach stage of an attack. It is designed to automate the "hands-on-keyboard" exploitation phase once a system has been compromised. In contrast, PENTESTAGENT addresses the entire penetration testing pipeline—from reconnaissance and vulnerability analysis to exploitation—providing a more comprehensive solution. This broader scope is critical for real-world scenarios, where early-stage tasks are just as vital as post-breach actions for assessing and improving security.

Enigma [93], on the other hand, extends the SWE-agent framework by integrating interactive tools that support solving Capture The Flag challenges. Its design primarily targets challenges in the crypto and reverse engineering domains and still relies on human-in-the-loop interactions. Although Enigma's interactive interfaces are effective for guiding the agent through specific problem domains, its reliance on manual intervention limits full automation. In contrast, PENTESTAGENT

is engineered to operate autonomously across diverse attack stages, reducing the need for human feedback and thus enabling more consistent performance in automated penetration testing.

Overall, these comparisons illustrate that while AutoAttacker and Enigma contribute valuable insights and capabilities, PENTESTAGENT distinguishes itself by offering an end-to-end automated solution.

3.5.2 Limitations on Performing Sophisticated Pentesting

Our system, PENTESTAGENT, focuses on exploiting individual vulnerable applications and services to help identify and mitigate these vulnerabilities. However, more sophisticated attack planning may be required in more complex penetration testing scenarios, such as red team simulations. These scenarios often involve combining several vulnerabilities to achieve a more challenging yet impactful exploit. For example, an SSRF vulnerability could be used as an intermediary step to exploit an internal application, eventually leading to obtaining root privileges.

While addressing such sophisticated attack strategies is beyond the scope of this chapter, our framework, PENTESTAGENT, can still be valuable in these complex scenarios. Our system can identify and validate exposed vulnerabilities, such as SSRF, which can serve as starting points for further exploitation. This initial identification and validation process can significantly contribute to the overall penetration testing workflow, providing a foundation upon which more advanced exploitation techniques can be built.

3.6 Conclusion

This chapter presents PENTESTAGENT, a novel LLM-based framework for automated penetration testing designed to address the limitations of existing frameworks: limited pentesting knowledge and insufficient automation. By leveraging a multi-agent architecture and incorporating various

LLM techniques like retrieval augmented generation and chain-of-thought, PENTESTAGENT enhances the penetration testing process through improved knowledge integration and automation.

Our comprehensive benchmark, based on VulHub's vulnerable Docker environments and HackTheBox CTF challenges, provided a comprehensive test bed of PENTESTAGENT. The evaluation results demonstrate that PENTESTAGENT achieves strong performance in task completion and overall efficiency.

3.A Appendix

3.A.1 Prompts

This section specifies more prompts used in the PENTESTAGENT pipeline.

The following prompt generates a structured output of the reconnaissance summary. Specifying the output structure and providing a comprehensive example guides the agent to output relevant information and reduces hallucination.

Reconnaissance Summary Prompt (Simplified)

Provide a summary of all reconnaissance findings ...
The summary of findings should be presented in valid JSON format with the following fields: {FORMAT SPEC.}
For example, {OUTPUT FORMAT EXAMPLE}

The following prompt summarizes the search results into a structured output for subsequent parsing and storing.

Search Results Summary Prompt

List ALL CVE numbers, URLs, keywords, and their applicable version relevant to exploit the vulnerabilities of {APP}. The results should be presented in valid JSON format with the following fields: {FORMAT SPEC.} ...

From our initial attempts, we found that the LLM is not familiar with software versioning. Therefore, we added a paragraph containing descriptions and examples to demonstrate how to handle software versions as few-shot learning. We use the following prompt to extract the desired information.

Exploit Procedure Analysis Prompt (Simplified)

RAG & CoT

Give a concise summary of the entire repository to answer the following questions:
1) whether this repository contains an exploit targeting a particular service or app;
2) What effect does the exploit have? Use one phrase to summarize the effect (e.g., remote command execution);
3) What relevant service/app version can this exploit be applied to?

Few-shot Learning

Note the app version is typically formatted as x.y.z. Explicitly state the version with the following formats ...
4) what are the requirements to run this exploit? (e.g., OS, library dependencies, etc.)

Structured Output

You should always respond in valid JSON format with the following fields: {FORMAT SPEC.} ...
For example, the response looks like this: {OUTPUT FORMAT EXAMPLE}

We designed the following prompt to generate a list of potential attack surfaces given a particular service or application.

Attack Surface Suggestion Prompt (Simplified)

List out all vulnerabilities ranked by confidence that can be used to exploit {app} {version} and provide the details about the vulnerabilities and the reasons to support each selection . . .
The details should include . . .
Make the selections by checking whether {version} is within the applicable version of the exploit and the vulnerability types . . .
The results should be presented in valid JSON format with the following fields: {FORMAT SPEC.} . . .
For example, {OUTPUT FORMAT EXAMPLE}

We designed the following prompt to generate a list of exploits for each potential attack surface.

Exploit Suggestion Prompt (Simplified)

List out paths of all relevant repositories ranked by the confidence that contain exploits . . . applicable to {app} {version} and provide the details about the exploit and reasons to support each selection . . .
The details should include . . .
Make the selections by checking whether {version} is within the applicable version of the exploit and the execution effects . . .
The results should be presented in valid JSON format with the following fields: {FORMAT SPEC.} . . .
For example, {OUTPUT FORMAT EXAMPLE}

After obtaining a list of needed information, the execution agent uses the following prompt to query the environmental information database to fill in the information.

Execution Information Query Prompt (Simplified)

Based on the known information, try to provide the information listed here. {INFO NEEDED . . . }

CoT

You should examine the information needed one by one. For each piece of information needed, you should . . .

RAG

You should use your query tool to learn about the target environment . . .

Structured Output

The results should be presented in valid JSON format with the following fields: {FORMAT SPEC.} . . .

3.A.2 Benchmark Construction

VulHub Benchmark Construction. We use CVSS and EPSS scores to determine the difficulty of exploiting vulnerabilities. CVSS provides a numerical score reflecting the properties of vulnerabilities. Since most of the CVEs on VulHub adopt CVSS version 3.x metrics, we use this as our reference to assign difficulty levels. The numerical score is made of two parts: exploitability and impact. For our penetration testing purpose, we use the exploitability metric as the reference to assign difficulty levels. The exploitability score reflects the ease and technical means by which the vulnerability can be exploited [94]. A higher exploitability score indicates that the vulnerability is easier to exploit. We studied the distribution of exploitability scores, as shown in Fig.3.11. We found that most exploitability scores are above 3.0, and exploitability scores of 2.0 and 3.0 make natural cutoffs for easy, medium, and hard difficulties. Some vulnerable applications or services have more than one CVE number. We select the CVE to use based on the EPSS score. The EPSS scores measure how likely a vulnerability will be exploited in the wild. A higher EPSS score indi-

cates the vulnerability is more likely to be exploited, making it more realistic for penetration tasks. Fig. 3.12 shows the distribution of the EPSS scores of the CVEs in our benchmark dataset.

In addition, we remove the Docker images that are not associated with a CVE number and do not have CVSS 3.x scores. Additionally, some vulnerable applications are removed from the dataset due to complicated setup processes, such as requiring a license key from a service provider. To maintain integrity and fairness in our evaluations, we strictly prohibit PENTESTAGENT from directly accessing any content from VulHub repository, thereby preventing any advantage or bias in our testing methodology.

Fig. 3.14 shows the difficulty rating distribution of our VulHub benchmark dataset.

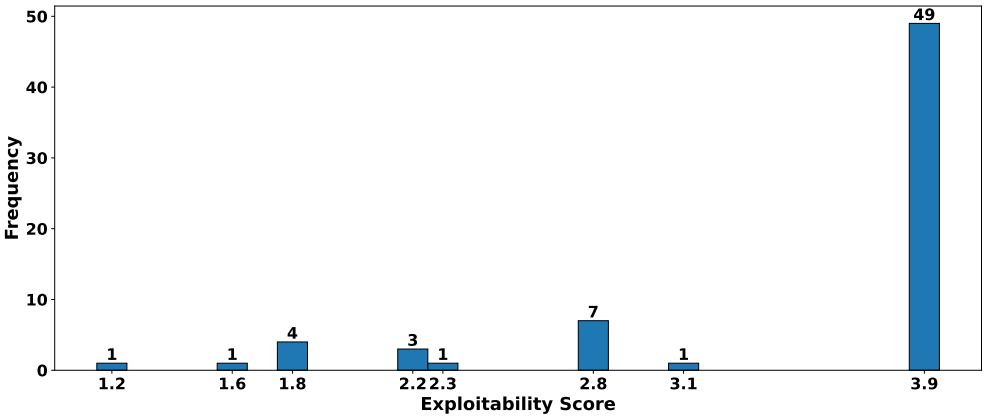


Figure 3.11: Distribution of exploitability scores

HackTheBox Benchmark Construction. In selecting HackTheBox (HTB) challenges for evaluating PENTESTAGENT, we aimed to create a diverse and representative set of targets that reflect real-world penetration testing scenarios. Our selection focused on key aspects such as operating system diversity, vulnerability relevance, and difficulty levels to ensure a comprehensive assessment.

To evaluate PENTESTAGENT 's adaptability across different environments, we included both

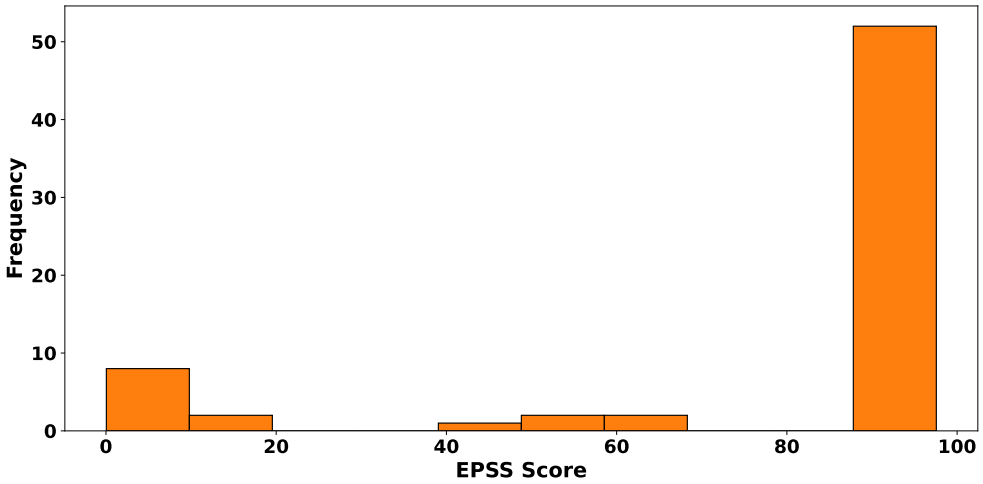


Figure 3.12: Coverage of EPSS scores

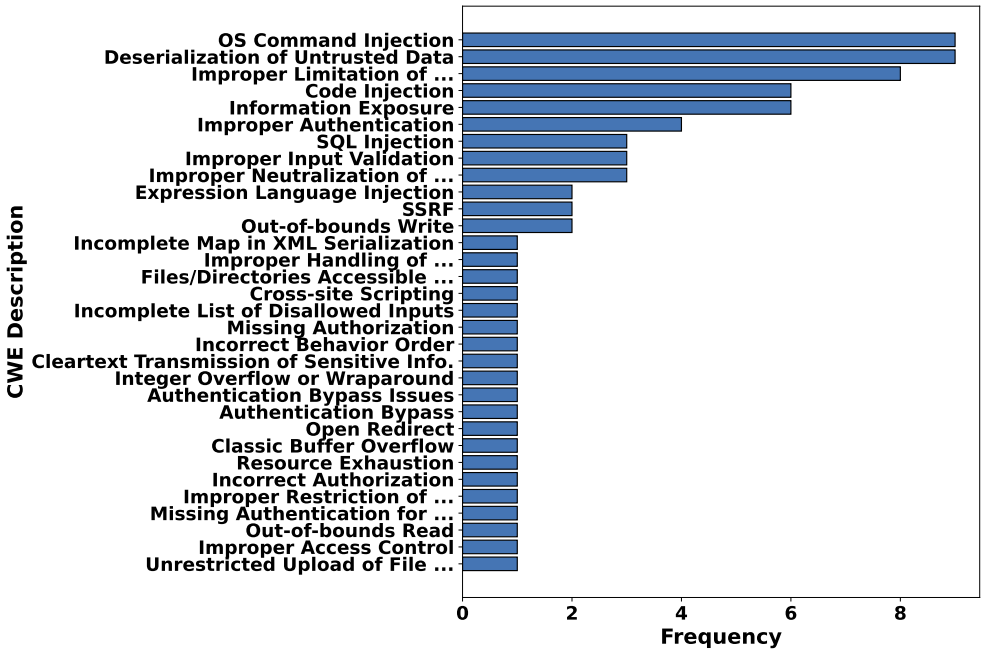


Figure 3.13: Coverage of CWE

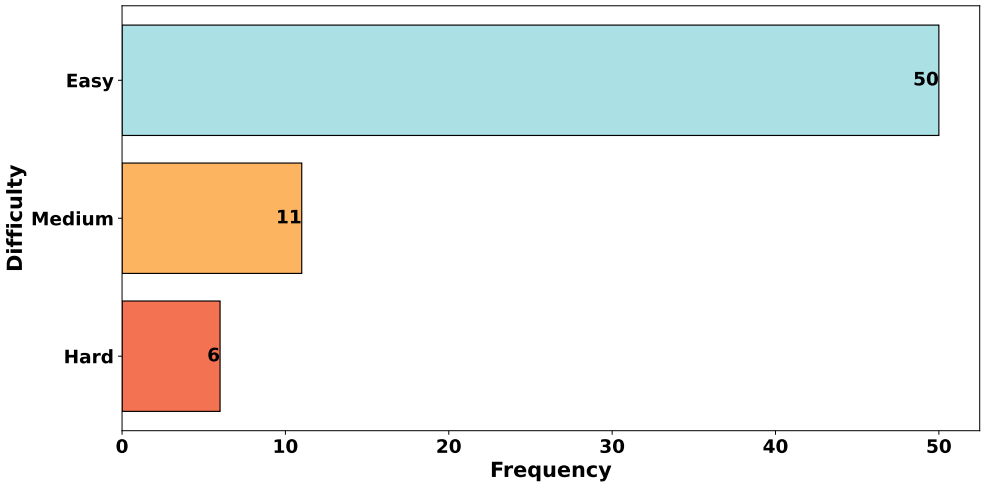


Figure 3.14: Distribution of exploitation difficulty ratings

Linux and Windows machines. This diversity reflects the variety of systems encountered in real-world engagements and ensures that PENTESTAGENT is tested across different exploitation techniques. The chosen machines also incorporate well-known vulnerabilities spanning a decade, allowing us to assess PENTESTAGENT 's ability to handle both historical and contemporary exploits. For instance, Blue features the EternalBlue vulnerability (CVE-2017-0144), a widely known Windows SMB exploit, while Legacy includes MS08-067 (CVE-2008-4250), another critical SMB-based attack.

We selected machines across easy, medium, and hard difficulty levels to analyze PENTESTAGENT 's performance in different attack scenarios. Easier challenges, such as Lame and Optimum, test fundamental exploitation techniques, while medium and hard machines, such as Stratosphere (CVE-2017-5638) and Reel (CVE-2017-0199), require deeper reconnaissance, multi-step attacks, and more advanced reasoning. This progression ensures that PENTESTAGENT is evaluated not only on basic automation tasks but also on its effectiveness in handling complex, real-world pentesting challenges.

This carefully selected set of HTB challenges, as shown in Table 3.3, allows for a thorough assessment of PENTESTAGENT 's automation capabilities, performance across different vulnerability types, and effectiveness in progressively complex penetration testing scenarios.

Table 3.3: PENTESTAGENT's performance among HackTheBox CTF challenges

Machine	**Difficulty**	**Completed Stage**
Sau	Easy	2/3 (I.G, V.A)
Pilgrimage	Easy	1/3 (V.A)
Lame	Easy	3/3
Topology	Easy	3/3
PC	Easy	3/3
Blue	Easy	3/3
Shocker	Easy	2/3 (V.A., E)
Optimum	Easy	3/3
Legacy	Easy	3/3
Stratosphere	Medium	2/3 (V.A., E)
Reel	Hard	2/3 (V.A, E.)

Table 3.4: PENTESTGPT's performance among HackTheBox CTF challenges

Machine	**Difficulty**	**Completed Stage**
Sau	Easy	2/3 (I.G, V.A)
Pilgrimage	Easy	1/3 (V.A)
Lame	Easy	2/3 (V.A., E)
Topology	Easy	2/3 (V.A., E)
PC	Easy	0/3
Blue	Easy	2/3 (V.A., E)
Shocker	Easy	0/3
Optimum	Easy	3/3
Legacy	Easy	3/3
Stratosphere	Medium	0/3
Reel	Hard	1/3 (I.G.)

Benchmark Coverage. Our evaluation was conducted using a benchmark dataset comprising known vulnerabilities, which raises questions about the practicality in real-world scenarios. Firstly, it is important to recognize that known vulnerabilities pose significant risks. Many organizations

and institutions struggle with timely patching practices, contributing to vulnerable and outdated components ranking 6th on the OWASP Top 10 Web Application Security Risks. [91] Additionally, while our benchmark dataset features known vulnerabilities, we selected environments based on their Exploit Prediction Scoring System (EPSS) scores. These scores reflect the likelihood of a vulnerability being exploited in real-world scenarios. The dataset's mean EPSS score is 79.58, with a median of 97.19, indicating that the vulnerabilities represented are highly likely to exist and be exploitable in practical settings. Moreover, finding open datasets containing zero-day or even one-day vulnerable environments remains challenging. By focusing on known vulnerabilities with high EPSS scores, our evaluation ensures that PENTESTAGENT operates within a realistic and credible context, assessing its effectiveness in addressing vulnerabilities that pose genuine risks to cybersecurity.

3.A.3 Additional Evaluation Results

The detailed pentesting performance of both systems on HackTheBox challenges are available in Table 3.3 and Table. 3.4.

In addition to the HackTheBox evaluation, we compare PENTESTAGENT and PENTESTGPT on VulHub targets, analyzing both completion levels and overhead, as shown in Fig. 3.15 and Fig. 3.16.

PENTESTAGENT significantly outperformed PENTESTGPT across all penetration testing stages. In intelligence gathering, PENTESTAGENT achieved an 80% completion rate, compared to only 10% for PENTESTGPT, demonstrating its superior ability to extract relevant target information. In vulnerability analysis, PENTESTAGENT completed 100% of the tasks, whereas PENTESTGPT again achieved just 10%, highlighting its limited capability in identifying and assessing vulnerabilities. During exploitation, PENTESTAGENT successfully completed 70% of tasks, more than

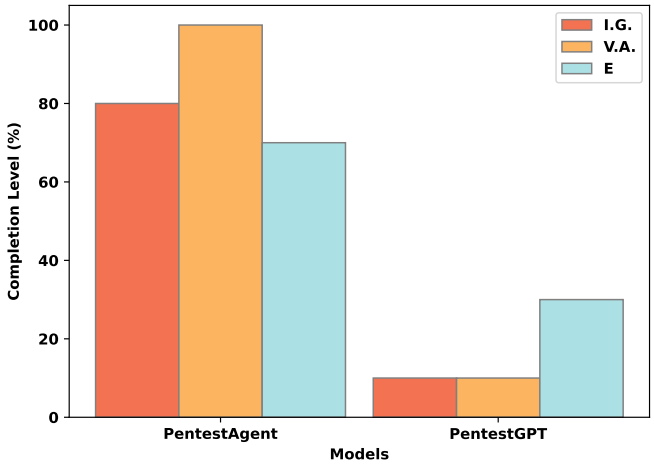


Figure 3.15: Comparison of completion levels on VulHub targets

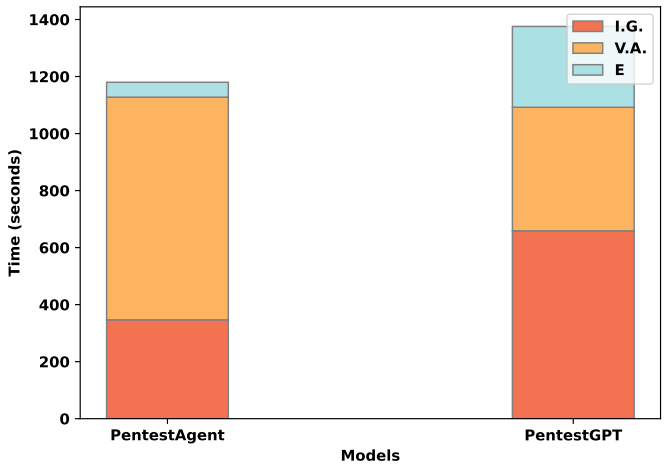


Figure 3.16: Comparison of average time spent on VulHub targets

double PENTESTGPT 's 30%, confirming its stronger execution capabilities.

Efficiency is crucial in penetration testing automation, and PENTESTAGENT consistently required less time than PENTESTGPT in key stages. It completed intelligence gathering in 212.9 seconds, whereas PENTESTGPT took 658.7 seconds, over three times longer. Similarly, in exploitation, PENTESTAGENT finished in 58.6 seconds, compared to 283.5 seconds for PENTESTGPT, demonstrating its more streamlined attack execution. While PENTESTAGENT took longer in vulnerability analysis (698.8 seconds vs. 433.5 seconds), this additional time contributed to its higher success rate, ensuring a more accurate and actionable assessment.

The results confirm that PENTESTAGENT is both more effective and more efficient than PENTESTGPT. It achieves higher completion rates across all stages, particularly in intelligence gathering and vulnerability analysis, which are critical for successful exploitation. Moreover, its lower overhead in intelligence gathering and exploitation makes it a more scalable and practical solution for real-world penetration testing. While its vulnerability analysis takes slightly longer, this trade-off results in more reliable and successful attack execution, solidifying PENTESTAGENT as a robust and efficient automated pentesting framework.

CHAPTER 4

AEAS: ACTIONABLE EXPLOIT ASSESSMENT SYSTEM

4.1 Introduction

The ever-growing volume of cataloged vulnerabilities poses a significant challenge for security practitioners: *how to effectively prioritize vulnerabilities for exploitation or remediation?* Within the vast pool of known vulnerabilities, only a small subset is practically exploitable. Identifying these high-impact cases remains difficult due to the lack of actionable signals. Although severity scoring systems exist, they primarily focus on theoretical risk rather than practical exploitability. This gap has led to inefficiencies in both offensive operations and vulnerability triage workflows.

This challenge becomes more severe at scale. Automated scanners and monitoring systems routinely report large numbers of vulnerabilities, many of which have public exploit artifacts available. However, the substantial quantity of available exploits is accompanied by wide variation in quality. This inconsistency complicates efforts to identify which vulnerabilities are realistically exploitable in a given context. In our empirical study detailed in Section 4.4.2, we manually examined over 600 exploit candidates associated with 47 vulnerabilities and found that only 51.4% were capable of achieving their intended impact. The remaining exploits were either non-functional, incomplete, or usable only as proof-of-concept (PoC) demonstrations. This disconnect between the presence of exploit artifacts and their actual utility underscores the limitations of current vulnerability assessment practices.

Current scoring systems provide limited support for this need. The Common Vulnerability Scoring System (CVSS) [89] and the Exploit Prediction Scoring System (EPSS) [90] are among

the most widely used. CVSS relies on manually assigned metrics that are often incomplete, or inconsistent across different reporters [95], [96]. These scores reflect static characteristics of vulnerabilities but fail to account for real-world exploitation evidence. As a result, CVSS scores are known to correlate poorly with actual exploitation in the wild [97], [98]. EPSS improves on this by incorporating external data sources and predictive modeling, but it lacks transparency, offering exploitation probability without evidence or reasoning led to the prediction. Both systems do not evaluate whether a concrete, runnable exploit exists or how usable it is in practice. As a result, practitioners often face a disconnect between high-severity scores and the operational reality of exploit availability. For instance, many vulnerabilities with high CVSS scores lack usable exploit code, while others with low scores may have functional exploits readily available in public repositories.

In parallel, penetration testing tools and vulnerability scanners have improved in indexing exploit-related information, but they rarely offer systematic prioritization grounded in exploit artifacts. Platforms [99], [100] such as Metasploit [74] and Nuclei [101] provide access to PoCs and exploit scripts. However, these tools often emphasize breadth over quality. Many listed PoCs are incomplete, difficult to execute, or require substantial modification before use. Moreover, few tools distinguish between verified exploits and scripts that merely reproduce conditions for triggering a vulnerability. As a result, practitioners must manually examine each artifact and determine whether it applies to their target environment. This process is labor-intensive and does not scale well.

Together, these limitations reveal two key gaps in the current vulnerability assessment landscape:

G1. Actionability. Practitioners lack systems that help identify and prioritize exploits that are truly ready for use. We define actionable exploits as those that are (i) available, (ii) func-

tional, and (iii) require minimal setup. Availability ensures that an exploit can be obtained and inspected. Functionality determines whether the exploit can achieve meaningful impact, such as remote code execution or privilege escalation. Minimal setup reduces friction by minimizing environment-specific dependencies or the need for manual configuration. Each of these dimensions is necessary for an exploit to be useful in automated red teaming, exploit development, or remediation prioritization. Existing tools and scoring systems do not evaluate these properties, resulting in mismatches between perceived and actual risk.

G2. Automation. Most workflows still rely heavily on manual effort to assess exploit actionability. Analysts must examine exploit code, cross-reference documentation, and test scripts under different configurations. Although some tools automate the retrieval of exploit artifacts, few support automated reasoning across heterogeneous, unstructured inputs such as blog posts, README files, code snippets, or configuration instructions.

Recent advances in large language models (LLMs) offer new opportunities for addressing these challenges. LLMs have demonstrated strong performance in tasks involving text summarization, contextual interpretation, and code analysis. These capabilities make them well-suited for analyzing unstructured security artifacts from sources such as public exploit repositories, and technical documentation. Early research has explored using LLMs to process unstructured security-related data like blogs and emails [102]. However, these efforts have generally focused on textual classification or summarization tasks and do not extend to exploit-level analysis. More importantly, they do not attempt to assess whether an exploit is available, functional, or easy-to-use.

In this chapter, we present AEAS, an automated and actionable exploit assessment system designed to address a critical and underexplored problem: identifying and prioritizing vulnerabilities based on the actionability of public exploit artifacts. Rather than replacing existing systems such as CVSS or EPSS, AEAS complements them by answering several operational questions: whether

a runnable exploit exists, whether it can be expected to work reliably in practice, and how difficult it is to deploy in real-world conditions.

A straightforward approach to this problem would be to execute each public exploit in a test environment and observe its behavior. However, this strategy is expensive, time-consuming, and difficult to scale. Many exploits require custom environments, manual configuration, or non-trivial setup. Large-scale execution also raises practical concerns, such as infrastructure cost, long test cycles, and difficulty maintaining a low profile in sensitive settings. As a result, we design AEAS to work through static analysis. Our goal is to approximate the benefits of dynamic analysis, understanding exploit feasibility and operational impact, without actually running the code.

This static analysis approach presents several technical challenges. Public exploit artifacts are noisy and highly inconsistent. Exploits often lack structured metadata and may be distributed alongside unstructured notes, blog posts, or incomplete documentation. To address this, AEAS includes a preprocessing pipeline that extracts contextual information, normalizes formats, and identifies relevant components for analysis. More importantly, exploit actionability cannot be captured through surface-level signals. It involves reasoning about various distinct features that contribute to exploit actionability, such as execution complexity and impact. We define a structured schema of such features and extract them using a hybrid approach that combines LLM-based reasoning and summarization with rule-based validation and aggregation.

Extracting these features accurately is not a trivial application of existing LLM capabilities. Off-the-shelf prompting often yields noisy or inconsistent outputs, particularly for technical security tasks. To improve robustness, we introduce task-specific prompt templates and chain-of-thought reasoning to guide LLMs through structured interpretation. Our ablation study shows that these techniques improve both accuracy and consistency across diverse model backbones. The extracted features are used to compute an exploit-level actionability score, which is then aggregated

into vulnerability-level rankings to reflect overall severity.

We construct a diverse evaluation pool of over 5,000 vulnerabilities published recently, covering a wide range of severities and exploit types. These vulnerabilities are linked to more than 600 applications frequently encountered by industry red teams, based on direct input from security professionals. We use AEAS to analyze this entire pool, demonstrating its scalability and broad applicability. We also conduct further manual verification and expert validation on representative subsets. The results show that AEAS achieves a 100% success rate in its top-3 exploit recommendations and produces vulnerability-level rankings that closely match expert judgments, validating both the effectiveness and practical utility of the system.

Contributions. This chapter makes the following contributions:

- We present AEAS, an automated framework for actionable exploit assessment. AEAS identifies high-quality exploits based on their functional and operational characteristics.
- AEAS introduces a task-specific analysis pipeline that combines large language models with rule-based heuristics to extract and reason over relevant features from unstructured exploit artifacts, enabling accurate analysis without requiring dynamic execution. To our knowledge, this is the first framework tailored for structured exploit assessment at scale.
- We evaluate AEAS on over 5,000 real-world vulnerabilities derived from commonly encountered applications in professional red teaming. The system achieves a 100% top-3 success rate in exploit recommendation and strong agreement with expert rankings, demonstrating its effectiveness and practical value.

4.2 Background and Related Work

4.2.1 Vulnerability Assessment

Vulnerability assessment is a critical process for identifying and prioritizing vulnerabilities within a system, especially when dealing with a large volume of assets. With numerous potential vulnerabilities across a network or environment, evaluating which ones pose the greatest risk is a complex task. Without a structured analysis, it is inefficient and impractical to address every vulnerability individually. Instead, vulnerability assessment provides a structured analysis from multiple perspectives to help testers prioritize vulnerabilities to exploit effectively. Over the years, several security metrics have been developed to assess the vulnerabilities [103], with the CVSS[89] and the EPSS[90] being the most widely adopted for system security assessments.

CVSS assigns a severity score ranging from 0.0 to 10.0 based on factors such as exploitability and impact. While widely used, CVSS suffers from significant issues of accuracy and consistency [96]. Wunder et al. [95] highlighted key shortcomings in a user study, pointing to ambiguous metric definitions and arbitrary manual scoring, often conducted without proper adherence to CVSS documentation. These inconsistencies undermine its reliability as a prioritization tool. A significant enhancement in CVSS v4.0 is the exploit maturity metric, which estimates the likelihood of a vulnerability being exploited based on factors such as exploit techniques, the availability of exploit code, and active "in-the-wild" exploitation [94]. Despite its potential value, this metric is rarely included in published CVSS scores, leaving a critical gap in practical vulnerability prioritization.

In response to these gaps, several external sources have been developed to provide insights into exploiting maturity. For example, EPSS [98] employs a predictive model to estimate the likelihood of a vulnerability being exploited. While EPSS introduces a useful predictive capability, it lacks

transparency in its outputs, making it difficult for practitioners to understand or validate the rationale behind the results. Several academic works [96], [104]–[108] aim to improve vulnerability assessment by leveraging machine learning techniques, such as NLP techniques [109] and neural networks [106]. However, like EPSS, these approaches still suffer from significant challenges related to transparency and explainability, limiting their practical utility in real-world scenarios.

Commercial vulnerability management tools also attempt to address these shortcomings by integrating exploit maturity assessments into their services. For instance, Tenable [99] offers a Vulnerability Priority Rating (VPR), providing a more nuanced and sophisticated metric for vulnerability prioritization. However, these tools have significant drawbacks: full access to their vulnerability databases and advanced assessment features often require costly subscriptions, creating barriers for smaller organizations or independent researchers. Moreover, the proprietary nature of these methods limits transparency, making it difficult to assess the reliability or fairness of their evaluations. Besides, Community-driven initiatives, such as CVEmap [100], aggregate information from public sources to improve accessibility. However, these projects tend to focus primarily on data collection without deeper analysis, which limits their effectiveness in supporting vulnerability prioritization.

These challenges underscore the need for an automated and explainable vulnerability assessment system. Such a system could improve both accuracy and consistency, while offering clear, transparent reasoning behind its outputs, providing practitioners with actionable insights backed by transparent decision-making processes.

4.2.2 Exploit Assessment and Recommendation

Effective exploit assessment is crucial for identifying which available exploits are suitable for executing attacks. When vulnerabilities are discovered, a wide range of exploits or proofs of

concept (PoCs) are often associated with them. However, not all of these exploits or PoCs are of equal quality or applicability. Sometimes, it claims to be an exploit but, in fact, a PoC and verse visa. Therefore, determining the usability of exploits is crucial. Effective exploit assessment should consider factors such as exploit reliability, execution feasibility, and contextual applicability to the target environment.

Metasploit [74] is a widely used exploitation framework that provides a collection of ready-to-use exploits. Metasploit includes an exploit ranking system that helps testers quickly identify practical attack methods. However, Metasploit has several limitations. Its repository contains a relatively small number of exploits, meaning it may not support newly discovered vulnerabilities in a timely manner. Additionally, the exploit rankings provided by Metasploit [110] lack transparency, offering no detailed explanation for why a particular exploit is ranked as more feasible or effective. Lastly, the exploit ranking only considers the reliability.

Community-driven resources, such as Nuclei [101], provide comprehensive repositories for vulnerability-related information. These platforms are known for their rapid updates and broad coverage of PoCs. However, they focus primarily on PoC availability. This lack of coverage on available exploits leaves penetration testers with the responsibility of manually searching for exploits, as well as evaluating and selecting the most suitable exploit, which can be time-consuming and error-prone.

Several academic works [109], [111] study the exploit code structure in attempt to gain insights. Suciu et al. [109] attempted to improve exploitability prediction by analyzing PoCs. While this approach is helpful for vulnerability prioritization, it does not provide a systematic method for assessing the usability of existing exploits, leaving a significant gap in the exploitation assessment process.

Given these limitations, there is a clear need for an automated, explainable exploit assessment

and recommendation system. Such a system would analyze various factors influencing exploit usability, including exploit characteristics, reliability, and environmental constraints, enabling it to rank exploits effectively.

4.2.3 LLM-based Data Analysis

Building on the need for efficient vulnerability and exploit assessment, large language models (LLMs) present a promising solution for analyzing unstructured data. LLMs have demonstrated exceptional capabilities in fields like finance [112] and healthcare [113]–[115], where their ability to process and summarize vast amounts of text and interpret domain-specific information has significantly improved decision-making processes. Similarly, vulnerability and exploit assessment can leverage LLMs to extract valuable insights from diverse unstructured data sources, including technical blogs, vulnerability disclosures, exploit code, and software documentation.

Initial research efforts [116] have explored the application of LLMs in this context. For instance, Ashiwal et al. [102] proposed a framework that utilizes LLMs to process unstructured data, such as blogs and emails, and generate structured outputs relevant to security analysis. While this work highlights the potential of LLMs in aiding penetration testing workflows, it is limited to text-based analysis and does not fully consider the variety of data formats encountered in penetration testing, such as code snippets, configuration files, or exploit repositories. Several works have achieved some initial successes in detecting vulnerabilities [117]–[119] from the source code and generating patches [120]. These successes make us think we can use LLM to fill the gap of exploit assessment.

Despite their potential, generic LLMs face significant challenges in effectively supporting vulnerability and exploit assessment tasks. First, LLMs do not inherently understand what specific features are relevant to penetration testers. Without clear guidance, LLMs may produce outputs

that lack actionable insights or omit critical details. Second, even when given instructions to focus on specific features, LLMs often struggle to extract relevant information effectively and efficiently. This is due to their reliance on pre-trained general knowledge rather than utilizing the provided external context optimally. For example, an LLM may incorrectly attempt to extract features from documentation when the answer lies in associated code, or vice versa. Such inefficiencies not only hinder the assessment process but can also lead to hallucination issues, where the model generates incorrect or misleading information.

Many LLM techniques can help address these challenges. Retrieval-Augmented Generation (RAG) [64] enhances LLMs by allowing them to utilize external data for generating responses. This technique involves three main stages: indexing, retrieval, and response synthesis. Initially, the dataset is indexed for efficient retrieval. Upon receiving a query, RAG retrieves relevant information from the indexed dataset and combines it with the original query before sending it to the LLM for response synthesis. The chain-of-thought (CoT) [76] technique significantly improves the ability of large language models to perform complex reasoning. By guiding the LLM to follow a logical sequence of steps, this method enhances the model's problem-solving capabilities. Role-playing [77] asks the LLM to impersonate an imaginary character, allowing the LLM to operate with clear objectives and boundaries, thereby enhancing their efficiency and effectiveness. Structured output techniques can save time spent on iterative prompt testing and ad-hoc parsing, reducing overall LLM inference costs and latency, as well as developers' effort. Additionally, structured outputs ensure smooth integration with downstream processes and workflows [79].

Using these techniques collectively, we propose the design of a customized data processing pipeline tailored to the specific needs of penetration testing. This pipeline leverages a combination of techniques to guide the LLM in extracting relevant features from various data sources, ensuring a more targeted and reliable analysis. By incorporating domain-specific instructions and seamlessly

handling multiple data formats, our approach enhances the precision and effectiveness of LLM-based vulnerability and exploit assessments. This advancement bridges the gap between general-purpose LLM capabilities and the specialized requirements of exploit assessment, contributing to more accurate, transparent, and actionable insights.

4.3 System Design

4.3.1 System Overview

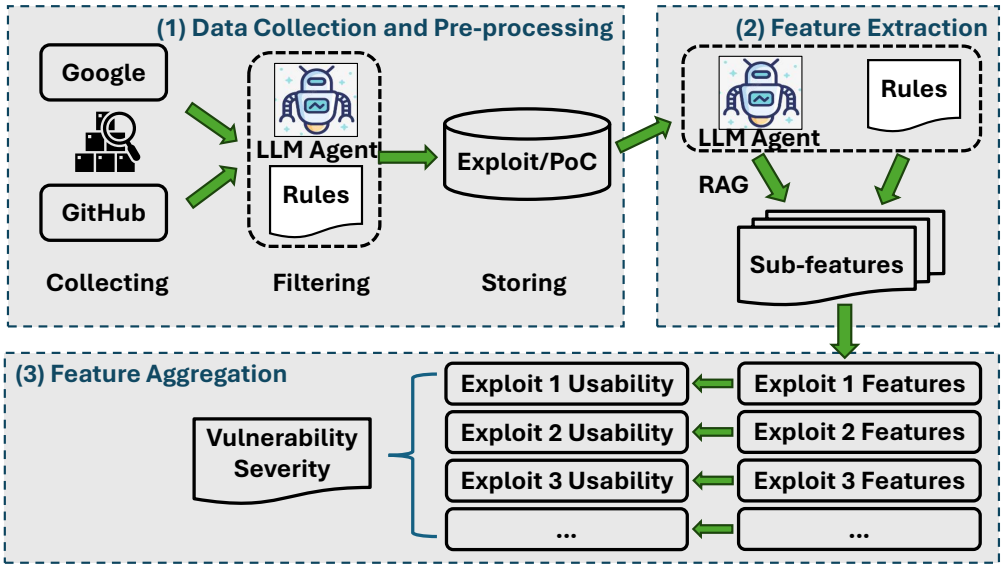


Figure 4.1: System Overview

As shown in Fig. 4.1, AEAS consists of three main stages: (1) Data Collection and Pre-processing, (2) Feature Extraction, and (3) Feature Aggregation. At the data collection stage, we collect data from designated data sources and apply both rule-based and LLM-driven filtering to facilitate more efficient processing in the subsequent stages. During the Feature Extraction stage, an LLM agent and rule-based heuristics are employed to extract sub-features, which are then combined to form comprehensive features for each exploit. Finally, in the Feature Aggregation stage,

these features are used to compute exploit exploitability scores, which are subsequently aggregated to determine overall vulnerability severity. We will further elucidate each stage in Section 4.3.2, 4.3.3, and 4.3.4, respectively.

4.3.2 Data Collection and Pre-processing

We collect vulnerability-related data from Google and GitHub, two widely used and complementary sources in real-world security workflows. Google enables access to web-based documentation, such as blog posts, tutorials, technical Q&A, and exploitation writeups that describe conditions for successful exploitation or mitigation. GitHub hosts a large volume of vulnerability-related repositories, including PoC scripts, exploit frameworks, and setup guides. Together, these sources provide both high-level contextual information and low-level technical artifacts relevant to vulnerability and exploit assessment.

However, data retrieved from these platforms is highly heterogeneous and noisy. Search results often include duplicated CVE summaries, advertisements, incomplete code templates, and repositories lacking functional content. These inconsistencies complicate downstream analysis and introduce irrelevant or misleading input.

To address these issues, we develop a hybrid pre-processing pipeline that combines rule-based heuristics with LLM agents. The rule-based component eliminates clearly uninformative or malformed data (e.g., binary files, enumerations of vulnerabilities, irrelevant metadata), while the LLM agent is used to analyze remaining content and filter out semantically trivial material, such as boilerplate summaries, generic explanations, and incomplete exploits.

This joint approach improves robustness against the diverse and unstructured nature of the input data. Rule-based filtering ensures efficiency and coverage for easily detectable artifacts, while the LLM enables semantic understanding across varied formats and documentation styles. Together,

these techniques allow us to retain relevant content and discard noise, ensuring that subsequent analysis modules operate on meaningful and actionable inputs. Further details of the data preprocessing steps are provided in Appendix 4.A.1.

Table 4.1: Key Features and Corresponding Sub-Features

Feature	Sub-Feature
Attack Vector	IsRemote
Attack Complexity	Info. Dependency, Attack Condition, Probability, UI, Privilege Req., Evasion
Impact	Code Exec., Privilege Escalation, Info. Leak, Bypass, DoS
Exploit Maturity	Relevance, Availability, Flexibility, Functionality
Popularity	# of Exploits, # of GitHub Stars & Forks

4.3.3 Feature Extraction

After completing data collection and preprocessing, we analyze each exploit along a structured set of features designed to capture its actionability. These features, summarized in Table 4.1, span five dimensions: attack vector, attack complexity, impact, exploit maturity, and popularity. Each dimension is further broken down into sub-features to support fine-grained assessment.

Our feature design is grounded in two foundations. First, we build upon the CVSS, which remains a widely used standard for assessing vulnerability severity. We adopt relevant CVSS metrics (e.g., attack vector and complexity) while refining others to better suit exploit-level analysis. For example, rather than using CVSS's impact triad (confidentiality, integrity, and availability), which may not directly reflect exploit intent or behavior, we introduce sub-features such as code execution and privilege escalation. These reflect concrete attacker outcomes and align more closely with red team workflows.

Second, we incorporate insights from experienced penetration testers in an industry red team with whom we actively collaborate. Drawing on their operational experience, we identified limitations in existing scoring systems and adjusted our feature definitions accordingly. For instance, the CVSS exploit maturity metric lacks specific guidance for distinguishing between PoC and fully functional exploits. Based on expert input, we introduce more granular sub-features that account for exploit completeness, required setup, and adaptability. We also add metrics such as exploit popularity which captures community adoption and reuse frequency, an aspect practitioners often consider but is missing from CVSS.

Together, these features are designed to reflect not only the theoretical severity of an exploit but also its real-world actionability, supporting both vulnerability triage and exploit selection tasks.

Attack Vector. This feature evaluates whether a vulnerability can be exploited remotely. Remote exploitation significantly broadens the attack surface by allowing attackers to target systems without physical or local network access, making it a crucial factor to consider in exploit assessment.

Table 4.2: Feature Values and Criteria

Feature	Value	Criteria
Attack Vector	Remote Not Remote	IsRemote = True Otherwise
Attack Complexity	Low High	Complexity score $>$ Complexity Threshold Otherwise
Impact	Code Exec. Privilege Escalation Info. Leak Bypass DoS	Code Exec. = True Code Exec. = False $\wedge$ Privilege Escalation = True Code Exec., Privilege Escalation = False $\wedge$ Info. Leak = True Code Exec., Privilege Escalation, Info. Leak = False $\wedge$ Bypass = True DoS = True
Exploit Maturity	None PoC Exploit	Relevance = False Relevance = True $\wedge$ ((Availability = False $\wedge$ Flexibility = False) $\vee$ Functionality = False) Relevance = True $\wedge$ (Availability = True $\vee$ Flexibility = True) $\wedge$ Functionality = True
Popularity	Low High	Popularity Score $>$ Popularity Threshold Otherwise

Attack Complexity. Attack complexity is assessed through six sub-features. Information de-

pendency considers whether prior knowledge (e.g., credentials) is needed. The attack condition evaluates whether specific configurations are required. Probability dependency examines reliance on uncontrollable factors, such as race conditions. User interaction determines whether user actions (e.g., uploading a file) are necessary. Privilege requirements assess the level of access needed, from none to administrative. Attack evasion checks for techniques to bypass detection. Together, these sub-features provide a detailed view of an exploit's prerequisites and difficulty.

Impact. Impact measures the potential consequences of a successful exploit, ranked by severity. The most critical is code execution, followed by privilege escalation (gaining unauthorized access), information leak (unauthorized data access), bypass (circumventing security mechanisms), and denial of service (disrupting system availability). Feature aggregation prioritizes the most severe potential impact.

Exploit Maturity. Exploit maturity measures the usability and sophistication of exploit code. This is determined through four sub-features. First, relevance evaluates whether the code directly relates to the vulnerability and can at least verify its existence. Second, availability measures whether the exploit code is accessible, such as being publicly hosted in repositories. Third, flexibility examines whether the code allows customization, such as modifying attack targets or goals. Finally, functionality assesses whether the exploit achieves goals beyond verification, such as remote code execution or bypassing authentication. These sub-features collectively provide insight into the practicality and versatility of available exploit code.

Popularity. Popularity is measured using community engagement metrics such as GitHub repository counts, stars, and forks. These metrics collectively indicate how widely a CVE topic is discussed and its relevance within the security community. Stars represent popularity and perceived value, while forks signal active engagement and adaptability. GitHub was selected as the primary data source due to its structured API and rich dataset, providing a reliable and efficient way to

assess trends. The popularity score can provide insights of the potential impact and prioritization of a vulnerability, as higher popularity often correlates with broader adoption and community validation.

Feature Extraction Query (Simplified)

Role-play

You're an excellent cybersecurity expert.

CoT

You should analyze the exploit from the following aspects with these steps: <Analysis Steps>

RAG

Here are some relevant documentation and code snippets: <Knowledge from RAG>

Structured Output

You should always respond in valid JSON format with the following fields: <Format Spec.>
For example, the response should look like this: <Output Example>

We utilize LLMs to extract all features except for popularity. To achieve this, we implement an RAG pipeline, which incorporates the collected data as contextual input during feature extraction. Our prompt design strategically employs LLM techniques, such as Chain-of-Thought (CoT) reasoning, to ensure accurate and comprehensive feature extraction. We show some examples of the detailed feature extraction outputs in Appendix 4.A.2.

For vulnerabilities without publicly available exploits, we assign a low default score. The lack of exploit data reflects the inherent difficulty in exploiting these vulnerabilities, as attackers lack the necessary information or tools to develop effective attacks.

4.3.4 Feature Aggregation

After extracting sub-features, we aggregate them to produce final values for the five key features: Attack Vector, Attack Complexity, Impact, Exploit Maturity, and Popularity. The aggregation involves either directly mapping values from sub-features or using criteria and thresholds to determine feature values, as summarized in Table 4.2. Finally, we compute an exploit exploitability score to represent the overall exploitability of an exploit and derive vulnerability-level severity by aggregating exploit scores. The aggregation process relies on several sets of weights, which serve as hyperparameters that can be adjusted to suit user needs. We determined the weights used in our experiments from expert elicitation to approximate values recognized by domain experts. We surveyed a panel of experienced pentesters who collaborated with us, asking them to rank the relative importance of features and sub-features. The weights are calculated from these rankings by averaging and normalizing them to approximate weights recognized by domain experts.

Attack Vector. The value of the Attack Vector feature directly corresponds to the IsRemote sub-feature. If the sub-feature indicates that the vulnerability can be exploited remotely (IsRemote = True), the attack vector is labeled as "True"; otherwise, it is labeled as "False."

Attack Complexity. Attack Complexity is derived from a weighted complexity score calculated using six sub-features: information dependency, attack condition, probability dependency, user interaction, privilege required, and attack evasion. Each sub-feature is assigned a weight based on its relative importance to the overall complexity. The weighted score is computed using the formula:

$$\textit{Complexity Score} = \sum_{i=1}^{6} w_i \cdot f_i \tag{4.1}$$

where w_i represents the weight of sub-feature i, and f_i represents the ordinal value of that sub-feature.

If the calculated score exceeds a predefined threshold, the attack complexity is categorized as "Low"; otherwise, it is categorized as "High."

Impact. The Impact feature is determined using a hierarchical evaluation of its four sub-features: code execution, privilege escalation, information leak, and bypass. Each sub-feature is assessed in descending order of severity, and the first applicable sub-feature defines the impact value. For instance, if Code Execution is true, it overwrites the other three sub-features, assigning "Code Execution" as the impact. In addition, DoS may be added to the Impact if the DoS sub-feature is true, regardless of the other four sub-features.

Exploit Maturity. This feature is determined by four sub-features: relevance, availability, flexibility, and functionality. Exploits are classified into three levels: None, PoC, and Exploit.None indicates the exploit is irrelevant and cannot verify the vulnerability. PoC applies when the exploit is relevant but fails to perform meaningful attack actions. Exploit is assigned when the exploit is relevant, functional, and either has accessible source code or supports flexible configuration changes (e.g., target, goal), enabling real-world use. This aggregation captures the distinction between trivial code samples and weaponized exploits.

Popularity. Popularity is derived from metrics such as the number of repositories, stars, and forks associated with the exploit. These values are combined into a Popularity Score using a weighted formula:

$$\textit{Popularity Score} = w_1 \cdot \textit{Repos} + w_2 \cdot \textit{Stars} + w_3 \cdot \textit{Forks} \tag{4.2}$$

Here, w_1, w_2, and w_3 denote the weights for the respective sub-metrics: number of repositories, stars, and forks. These weights are chosen based on the relative importance of each metric in determining popularity. If the computed score surpasses a designated threshold, the exploit's popularity is categorized as "High"; otherwise, it is labeled as "Low."

Exploit Actionability Score. We calculate the actionability score for the exploit by normalizing

the weighted sum of all feature scores.

$$\textit{Total Score} = \alpha_1 \cdot AV + \alpha_2 \cdot AC + \alpha_3 \cdot I + \alpha_4 \cdot EM + \alpha_5 \cdot P \tag{4.3}$$

$$\textit{Act. Score} = \|\textit{Total Score}\| \tag{4.4}$$

where AV, AC, I, EM, and P are the numerical representations of Attack Vector, Attack Complexity, Impact, Exploit Maturity, and Popularity, respectively. $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ are weights assigned to each feature. This score provides a comprehensive measure of an exploit's potential impact and feasibility. Intuitively, the actionability of exploits of a certain vulnerability can also reflect the severity of the vulnerability. To calculate the vulnerability severity, we aggregate the exploit scores for all associated exploits using the maximum value:

$$\textit{Vulnerability Severity} = \max_j(\textit{Act. Score}_j) \tag{4.5}$$

where j indexes all exploits related to a specific vulnerability. The hierarchical feature aggregation process transforms exploit-level sub-features into actionable vulnerability-level metrics, providing a solid foundation for automated and explainable vulnerability and exploit assessment. At the same time, this process can mitigate potential errors from LLM hallucinations, By considering all associated exploits, inaccuracies in individual exploit features are averaged out, ensuring robust and reliable assessments.

AEAS Output (Simplified)

Severity Score

The severity score of the vulnerability is: <Severity Score>

Exploitability Scores

The exploitability scores of the exploits are:

- Exploit 1: <Exploit 1 Score>
- Exploit 2: <Exploit 2 Score>
- ...

Exploit Features

Exploit 1:

- Attack Vector: <Attack Vector>
  - <Justification>

  [itemsep=2pt, parsep=2pt, topsep=0pt]
- Attack Complexity: <Attack Complexity>
  - <Justification>
- ...

Exploit 2: ...

As shown above, AEAS eventually provides a two-level output for each vulnerability: (1) a vulnerability-level severity score, computed from the highest-ranked associated exploit; and (2) a set of exploit-level exploitability scores and feature values, including attack vector, complexity, impact, maturity, and popularity, each accompanied by textual justifications generated via an LLM. These explanations summarize evidence extracted from exploit code and documentation. The intent is to help users understand not only *what* scores are assigned, but *why* those scores arise, offering transparency for both decision support and learning.

4.4 Evaluation

We evaluate our system to answer the following research questions:

RQ1. Exploit Actionability (Section 4.4.2). Can AEAS accurately identify actionable exploits? This question focuses on the exploit-level output of the system. We assess whether AEAS can identify effective exploits effectively by comparing its exploit recommendations against ground truth obtained through manual analysis.

RQ2. Vulnerability Severity (Section 4.4.3). Can AEAS assign severity scores that support effective vulnerability prioritization? We compare AEAS's vulnerability severity scores with those of EPSS using statistical agreement analysis and expert validation to determine whether AEAS offers improved support for practical decision-making.

RQ3. Ablation Study (Section 4.4.4). How do different LLM backbones and prompting strategies affect feature extraction quality? We examine whether alternative LLMs and prompt designs preserve feature extraction accuracy and consistency.

4.4.1 Evaluation Setup

4.4.1 Dataset

To evaluate AEAS, we constructed a large and diverse dataset of vulnerabilities derived from real-world penetration testing scenarios. Specifically, we collaborated with industry red team professionals, who provided a list of 655 frequently encountered applications based on their operational experience. We aggregated all known CVEs associated with these applications, resulting in a pool of more than 5,000 vulnerabilities.

Figure 4.2 shows the distribution of CVEs across these applications. While most applications

are associated with only a few vulnerabilities, approximately 15% (97 out of 655) have more than ten, illustrating the scale and prioritization challenge practitioners face in real environments.

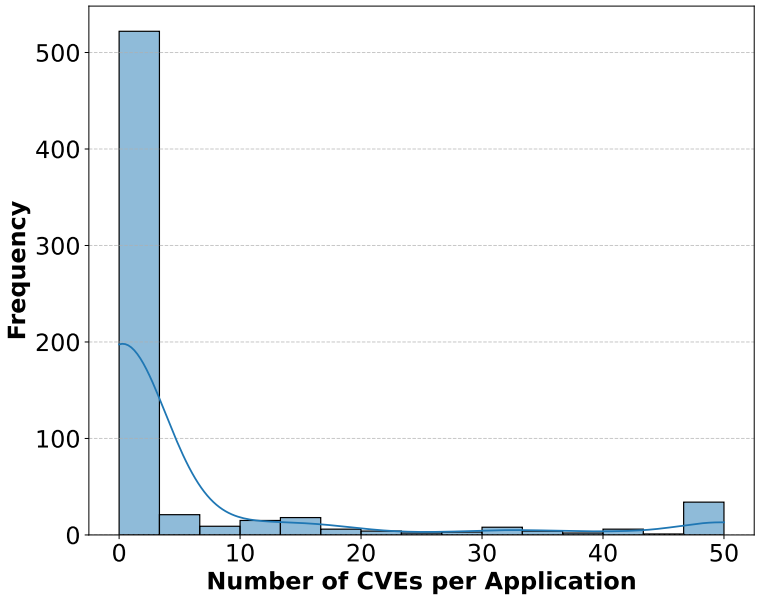


Figure 4.2: Distribution of Number of CVEs per Application

Due to cost and runtime constraints, not all vulnerabilities were used in further in-depth evaluation. Instead, we selected representative subsets of this dataset for different evaluation tasks. These subsets were incrementally sampled to maximize coverage across the following dimensions:

- **Vulnerability Types:** We prioritized maximizing coverage across Common Weakness Enumerations (CWEs) to capture a broad range of technical weaknesses and attack scenarios.
- **Severity Levels:** We ensure balanced coverage across CVSS and EPSS severity ranges to test how well AEAS handles vulnerabilities of varying impact and likelihood.
- **Mitigation of Prior Knowledge Bias:** To reduce the likelihood that LLMs rely on memorized information from their training data, we focused on vulnerabilities published on or after January 2024. This choice reflects a balance between two goals: mitigating prior knowledge bias, since

most models used in our system have training cutoffs in late 2023 (see Table 4.3), and ensuring sufficient availability of public exploit artifacts for analysis. While this strategy does not fully eliminate the possibility of memorization, it significantly reduces the influence of prior exposure and allows us to better assess the system's capability to extract features and reason about exploit actionability based on the inputs alone.

This evaluation setup enables a rigorous and realistic assessment of AEAS 's performance, while ensuring the dataset reflects both practitioner needs and research goals.

4.4.1 Actionability Metrics

To establish a objective standard for evaluating actionability of exploits, we define a set of quantitative metrics. These metrics capture various aspects of the exploit process:

- **Exploit Maturity:** Whether the exploit achieves its intended objective (e.g., code execution, privilege escalation) or it can only serve as a scanner to verify the vulnerability exists. This metric evaluates the practical viability of an exploit.
- **Completion Time:** The duration required to assess and execute the exploit, starting from its initial inspection and ending with either successful exploitation or a failure conclusion. This metric reflects the efficiency of the exploitation process.
- **Number of Errors:** The total number of errors or adjustments needed during the exploitation process. This includes syntax corrections, misconfigurations, and dependencies, providing insight into the exploit's ease of use.

Together, these metrics provide a comprehensive view of exploit actionability.

4.4.1 Environment Setup

To verify the actionability of exploits, we need to reproduce the vulnerable environments and use the exploits to attempt to exploit these vulnerable environments. To provide enough isolation for the safety consideration so that our simulated attacks do not cause any unintended damage, we built two virtual machines to serve as the attacker and victim machines. The simulated vulnerable environments are hosted on a virtual machine with 2 CPU cores and 8 GB RAM, running Ubuntu 22.04 LTS. To avoid interference with the testing process, we have disabled all irrelevant services that allow IO communication, such as SSH. The attacker machine is also hosted on a virtual machine with 16 CPU cores and 16 GB RAM, running Kali Linux 2024.1. The victim machine and the attacker machine maintain network connectivity via NAT. This setup ensures the attacker can simulate real-world network conditions when attempting to exploit the vulnerabilities while keeping isolated to the outside Internet. The assessments are run 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 evaluate our framework using a mix of general-purpose and reasoning LLMs. Table 4.3 summarizes their key properties.

Table 4.3: Summary of LLM Models

Model	Context Window	Knowledge Cutoff
GPT-4o-mini	128k tokens	Oct 2023
o3-mini (Reasoning)	200k tokens	Oct 2023
DeepSeek-V3-0324	128k tokens	Mar 2025
DeepSeek-R1 (Reasoning)	128k tokens	Feb 2024
Kimi-v1	128k tokens	Unknown
Kimi-k1.5 (Reasoning)	128k tokens	Unknown
Llama-3.3-70B-Instruct	128k tokens	Dec 2023

4.4.2 RQ1. Exploit Actionability

To evaluate the actionability of AEAS, we conducted a detailed manual verification of exploit artifacts to verify the quality of individual exploits. In this context, exploit actionability refers to the ability of a candidate exploit to achieve its intended goal, such as remote code execution or privilege escalation, with minimal configuration or manual intervention. By manually validating each exploit's functionality, runtime behavior, and execution conditions, we can measure how well AEAS identifies and prioritizes actionable exploits in practice.

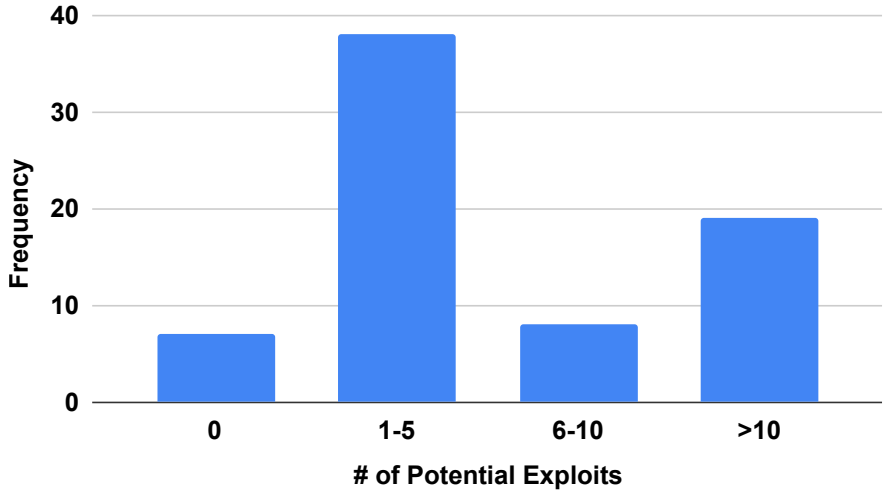


Figure 4.3: Distribution of Number of Exploits per CVE

Since the exploit-level manual analysis is time-consuming, we focused on a testing set of 71 vulnerabilities. The testing set was randomly and incrementally sampled from the dataset, ensuring that it included a diverse range of vulnerabilities with varying severities. Fig. 4.3 presents a histogram showing the distribution of the number of repositories associated with each vulnerability. Among these vulnerabilities, 7 had no available exploits, 37 had between one and five, 8 had between five and ten, and 19 had more than ten. This distribution underscores the need for effective prioritization mechanisms, as practitioners must often navigate a wide range of exploit options

with varying degrees of quality.

For the 64 vulnerabilities with available exploits, we manually evaluated a total of 611 potential exploits in the experimental environments described in Section 4.4.1.3, using the actionability metrics defined in Section 4.4.1.2. The evaluation process involved a consistent evaluator who attempted each exploit, recording to what degree it succeeded in achieving the intended functionality and the time required to achieve success. The evaluator also noted the number of errors encountered during the exploitation process. This manual evaluation provided a comprehensive understanding of the actionability of each candidate, allowing for a detailed analysis of the results. The evaluation revealed that 36 (5.9%) of the exploits lacked source code or executable files, providing only descriptive documentation. 115 (18.8%) exploits were usable as PoCs to verify the existence of vulnerabilities but could not achieve the desired objectives, such as remote code execution. Only 314 (51.4%) of the evaluated exploits successfully achieved their intended functionality. The remaining 182 (29.8%) exploits were either non-functional or failed to achieve the intended functionality, indicating a significant number of low-quality or incomplete exploits in the dataset. Fig. 4.4 shows an overall distribution of exploit maturity, and Fig. 4.5 presents a detailed view for CVEs with three or more exploits. This further underscores the substantial workload required to assess and prioritize the available exploits. At the vulnerability level, 47 (66.2%) of the 71 analyzed vulnerabilities had at least one functional exploit, while 54 (76.1%) had at least a PoC-level exploit available. However, 17 (23.9%) of the vulnerabilities lacked any functional exploit or PoC, further highlighting the variability in exploit availability and quality. On average, the successful exploits took 8.5 minutes to complete, while the PoC-level exploits took 9.0 minutes. The number of errors encountered during the exploitation process averaged 1.1 for successful exploits and 2.1 for PoC-level exploits.

We use these manual evaluation results as the ground truth to assess the performance of AEAS's

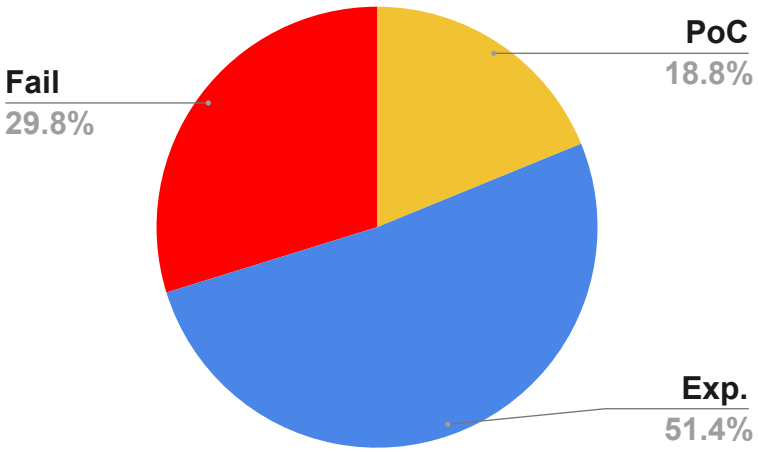


Figure 4.4: Overall Distribution of Exploit Maturity

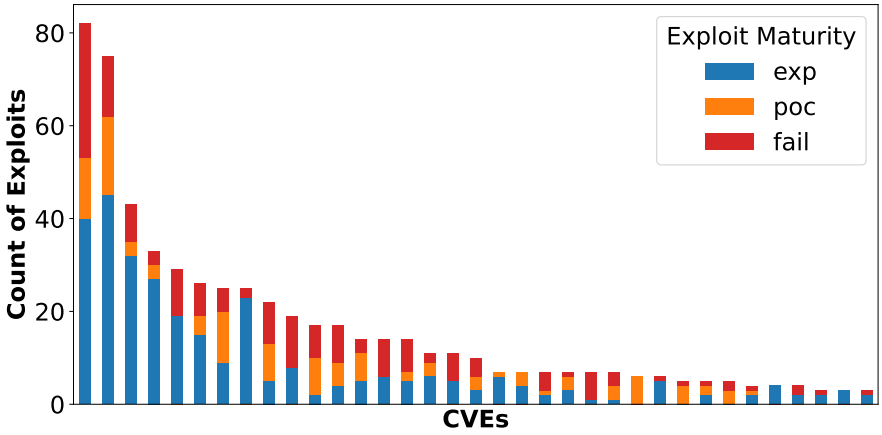


Figure 4.5: Distribution of Exploit Maturity per CVE

actionability ranking. We use the 47 vulnerabilities that have at least one functional exploit as the dataset. For each vulnerability, AEAS generated an actionability ranking based on scores computed using Equation 4.4. We compared these rankings with manually verified rankings to assess AEAS 's performance. Three key metrics were used to evaluate the accuracy and robustness of AEAS in this evaluation:

- **Top-k Success Rate**, which measures whether at least one exploit achieving the intended functionality appears in the top-k recommendations;
- **Precision@k**, which evaluates whether the highest-quality exploit (as determined by manual verification) is included in the top-k recommendations;
- **Recall@k for Top-j**, which assesses whether at least one of the top-j manually verified exploits is included in the top-k recommendations.

Table 4.4: Exploit Actionability Evaluation Results

Metric	**AEAS**	**Random Select**
Top-1 Success Rate	80.9%	51.4%
Top-3 Success Rate	100%	-
Precision @ 3 Rate	66.0%	-
Recall @ 3 for Top-3 Rate	78.7%	-

Table 4.4 summarizes the results of these evaluations. The results demonstrate that AEAS effectively prioritizes high-quality exploits. The Top-1 Success Rate shows that 80.9% of the time, AEAS ranks a successful exploit achieving the intended functionality as its top recommendation. Furthermore, the Top-3 Success Rate indicates that for all evaluated vulnerabilities, at least one functional exploit is included within the top three recommendations, achieving a perfect success rate of 100%. The results also show that AEAS outperforms random selection, which only achieves a Top-1 Success Rate of 51.4%. Besides, Precision@3 reveals that in 66.0% of cases, the

highest-quality exploit identified through manual evaluation is among AEAS 's top three recommendations. Finally, the Recall@3 for Top-3 metric shows that 78.7% of the top three manually verified exploits are included within the top three recommendations of AEAS.

These findings validate AEAS 's accuracy and effectiveness in exploit actionability assessment. AEAS reliably identifies and prioritizes high-quality exploits among exploits with various qualities. This not only reduces the time and effort required for manual evaluation but also ensures that practitioners can focus on the most actionable and impactful exploits for their analyses.

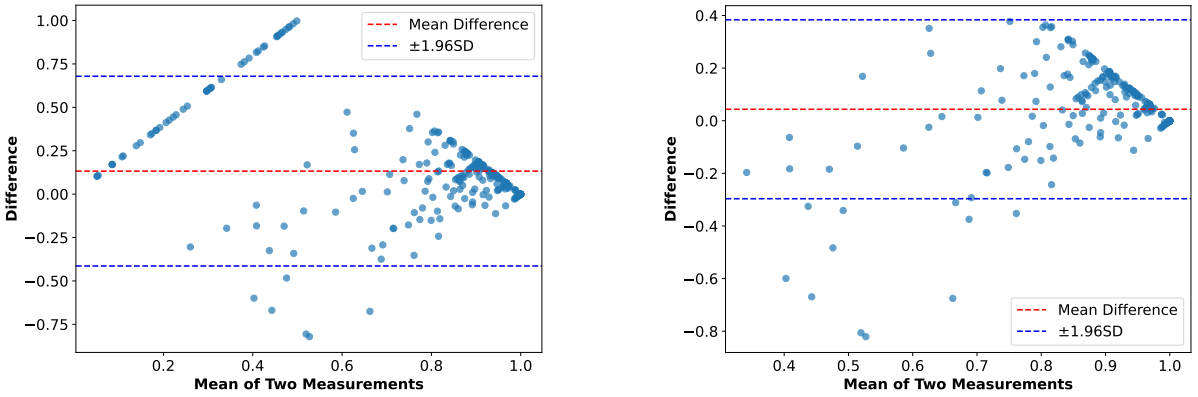
4.4.3 RQ2. Vulnerability Severity

We now evaluate AEAS at the vulnerability level to assess how well it supports prioritization decisions when aggregated across multiple exploit candidates. For this evaluation, we selected a representative subset of 292 vulnerabilities and over 1,000 associated exploits using the approach mentioned in Section 4.4.1.1. While AEAS primarily performs analysis on the exploit level, many security workflows require vulnerability-level severity scores to drive triage, remediation, or risk-based decision-making. To assess the effectiveness of AEAS in this context, we compare its vulnerability severity scores against those generated by EPSS.

EPSS is the most suitable existing baseline for this comparison because it is designed to predict real-world exploitation likelihood based on data-driven modeling. Unlike CVSS, which reflects only theoretical severity and lacks empirical grounding, EPSS incorporates external evidence such as exploitation trends and threat intelligence feeds. Although EPSS does not analyze exploit artifacts directly, it provides the strongest available benchmark for evaluating prioritization effectiveness in operational settings.

We conduct an agreement analysis to quantify how well AEAS's vulnerability scores align with those from EPSS. Agreement between the two can indicate that AEAS captures similar operational

signals through artifact-level analysis, while disagreement may suggest that AEAS offers new insights grounded in practical exploitability. This analysis allows us to examine both consistency and divergence, helping us understand whether AEAS produces reasonable vulnerability-level assessments that can serve as a practical complement to existing scoring systems.



(a) All Vulnerabilities

(b) Vulnerability with Available Exploits

Figure 4.6: Bland-Altman Plots for Vulnerability Severity Comparison

Agreement Analysis. We evaluate the agreement between AEAS's vulnerability severity scores and EPSS using Bland-Altman analysis, a statistical method that quantifies how closely two sets of measurements align and highlights systematic differences or outliers. Fig. 4.6 shows two Bland-Altman plots comparing AEAS's results with EPSS. Each point in the plot represents a vulnerability, where the x-axis denotes the average of two scores (i.e., AEAS and EPSS), and the y-axis denotes the difference between them. Agreement is considered strong when most differences are small and evenly distributed across the score range. For the full dataset, the mean difference between EPSS and AEAS is 0.13. A total of 32 vulnerabilities (10.96%) fall outside the 95% limits of agreement (defined as ±1.96 standard deviations from the mean), meaning that 89.04% of vulnerabilities show reasonable agreement.

We observe a clear trend in the lower-severity range as shown at the left side of Fig. 4.6a. For some vulnerabilities, AEAS consistently assigns lower severity scores than EPSS, especially when

no usable exploit is available. In such cases, AEAS assigns a score of 0.0, while EPSS may still assign moderate scores. This reflects a key difference in design: AEAS penalizes vulnerabilities without actionable exploit artifacts, while EPSS relies on statistical signals that may not account for the absence of usable code.

To better isolate the impact of exploit availability, we repeat the analysis on the subset of vulnerabilities that have at least one associated exploit. After removing the 53 vulnerabilities (19.15% of the dataset) without any exploit data, agreement improves substantially. As shown in Fig. 4.6b, the mean difference narrows to 0.04, and only 13 vulnerabilities (5.44%) fall outside the 95% limits of agreement.

We also observe a second pattern in the high-severity range towards the right side of the plots. When both EPSS and AEAS assign relatively high scores, AEAS tends to produce slightly lower values. This suggests that AEAS differentiates more sharply among high-risk vulnerabilities, assigning lower scores to those that are difficult to exploit despite having high theoretical severity. In contrast, EPSS tends to assign elevated scores more uniformly across this range, possibly due to correlations with metadata features rather than exploit-specific evidence.

These observations highlight a fundamental distinction between the two systems. EPSS provides a statistical estimate of exploitation likelihood using external signals and historical data, while AEAS grounds its assessment in direct analysis of exploit artifacts. As a result, AEAS is less likely to overestimate the risk of vulnerabilities that are severe in theory but impractical to exploit due to missing, incomplete, or unusable exploit code.

Outlier Analysis. To better understand the discrepancies between AEAS and EPSS, we conducted a detailed case study analysis of outliers identified in the Bland-Altman analysis. Specifically, we analyzed 36 cases with score differences exceeding one standard deviation among vulnerabilities with associated exploits. These outliers fall into two primary categories: vulnerabilities where

AEAS assigned higher severity than EPSS and those where it assigned lower severity.

Table 4.5: Outlier Analysis Summary

Case	**# of Case**	**Accuracy**
AEAS $\gg$ EPSS	9	66.7%
AEAS $\ll$ EPSS	27	100%
Overall	36	91.7%

Case 1: AEAS determines the vulnerability has a higher severity than EPSS. In this category, AEAS assigned higher severity to 9 cases (25%). Among these, three vulnerabilities had working exploits that achieved the attack goals in our experimental environment. EPSS likely assigned lower scores due to factors such as the complexity of exploitation, including kernel-level attacks or evasion techniques. However, AEAS correctly identified automated exploit scripts for these cases, which significantly simplified exploitation, demonstrating its ability to accurately capture actionability in such scenarios. Another three cases involved only PoC-level exploits, where the discrepancy arises from EPSS's underestimation of the severity of these vulnerabilities. Although these exploits were not fully functional, they provided sufficient information to demonstrate the vulnerability's existence and potential impact. AEAS's higher scores in these cases reflect its focus on practical actionability rather than theoretical severity. The remaining three cases were misclassified by AEAS. One exploit only provided a binary payload, which AEAS could not analyze; another exploit had comprehensive documentation but failed in our manual testing. The last case involved a honeypot exploit that was incorrectly treated as functional by AEAS. These instances represent limitations of our static analysis approach but constitute a small fraction of the dataset.

Case 2: AEAS determines the vulnerability has a lower severity than EPSS. In this category, AEAS assigned lower severity scores than EPSS to 27 cases (75%). Of these, 12 vulnerabilities did not have any exploit or PoC available through our online search, which naturally led to

lower scores under AEAS's exploit-centric assessment. The remaining 15 cases shared common characteristics: limited availability of exploit candidates, unclear or missing README files, and minimal code comments, all of which hindered AEAS's ability to analyze the code effectively. Additionally, most cases exhibited score differences close to one standard deviation, which aligns with AEAS 's more balanced score distribution compared to EPSS's heavily skewed distribution. Among these 15 cases, six had non-functional exploits upon manual verification, supporting AEAS 's assessment. Another five cases were associated with PoC-level exploits that EPSS scored relatively high, despite their limited functionality. The final four cases involved functional exploits with low actionability due to unclear documentation or manual execution requirements, which AEAS penalized.

Overall, the outlier analysis demonstrates that AEAS provides explainable results for vulnerability severity assessment. This explainability enables AEAS to justify instances where its evaluations diverge from EPSS, supported by exploit-level evidence. These cases demonstrate that AEAS can assess vulnerabilities more accurately than EPSS. By aggregating exploit actionability metrics into vulnerability severity scores, AEAS offers a reliable and interpretable framework that aligns well with established benchmarks while addressing critical gaps in exploit-level analysis.

Expert Validation. To further evaluate the accuracy and practical relevance of AEAS 's vulnerability severity scores, we conduct an expert validation study with six experienced security professionals from our collaborating institution. Importantly, these experts were not involved in the design, development, or tuning of AEAS. This separation ensures that their feedback is independent and not influenced by prior knowledge of the system's internal logic or feature choices, thereby reducing potential bias in the evaluation.

To make the manual validation manageable, we randomly selected 150 vulnerabilities from the larger pool of 5,000+ vulnerabilities analyzed by AEAS. Each expert reviewed a set of 25 vulner-

abilities and was shown anonymized, normalized severity scores produced by AEAS, CVSS, and EPSS. For each score, they indicated whether it was *Overrated*, *Underrated*, or a *Match* based on their experience with similar cases. This expert feedback allows us to directly compare AEAS 's assessments against professional expectations and complements the manual exploit-level validation in Section 4.4.2.

As shown in Table 4.6, AEAS aligns with expert judgment in 91.3% of cases, significantly outperforming EPSS (45.3%) and CVSS (53.3%). Notably, both EPSS and CVSS exhibit a much higher rate of Overrated assessments (53.3% and 46.0%, respectively) compared to only 6.0% for AEAS. This suggests that existing scoring systems tend to assign higher severity levels than practitioners find appropriate in practice, while AEAS produces scores that are more conservative and consistent with operational expectations.

Table 4.6: Accuracy of Vulnerability Severity Assessment

Dataset	**AEAS**	**EPSS**	**CVSS**
Match	137 (91.3%)	68 (45.3%)	80 (53.3%)
Overrated	9 (6%)	80 (53.3%)	69 (46%)
Underrated	4 (2.7%)	2 (1.3%)	1 (0.7%)

Further examination of the Overrated cases from AEAS revealed that the discrepancies were typically not due to errors in feature extraction. Instead, experts indicated that while the technical conditions of the exploit were correctly captured, the overall impact should be reflected in the score. For example, CVE-2023-31419 and CVE-2023-27704 were marked as overrated because their exploitation leads only to denial-of-service (DoS), which experts considered less severe than vulnerabilities enabling remote code execution. These observations underscore the inherent subjectivity in severity assessment, where impact interpretation may vary based on deployment context. Even in such cases, AEAS 's detailed feature-level output still provided valuable and accurate technical signals, offering an objective basis for expert interpretation.

For the few Underrated cases (2.7%), experts noted that mature exploits were missing from the input set used by AEAS. For instance, CVE-2017-3506 is known to have reliable exploitation pathways, but these were not downloaded by our system during data collection. This limitation reflects the dependency of AEAS on comprehensive exploit discovery, which in rare cases may omit relevant artifacts due to incomplete coverage or metadata inconsistencies.

Overall, this study demonstrates that AEAS produces vulnerability severity assessments that closely align with expert intuition, while maintaining transparency and interpretability through its evidence-driven design.

Table 4.7: Ablation Study on LLM Backbones

Metric	**GPT-4o-mini**	**o3-mini**	**DeepSeek-V3**	**DeepSeek-R1**	**Kimi-v1**	**Kimi-k1.5**	**Llama-3.3**
Avg. Accuracy	64.3%	78.7%	76.0%	83.3%	62.6%	67.1%	70.5%
Avg. Variance	19.6%	14.7%	17.0%	16.7%	22.7%	21.4%	18.6%

Table 4.8: Ablation Study on LLM Techniques

Model	**Metric**	**w/ CoT**	**w/o CoT**
GPT-4o-mini	Avg. Accuracy	64.3%	59.4%
	Avg. Variance	19.6%	22.6%
o3-mini	Avg. Accuracy	78.7%	73.3%
	Avg. Variance	14.7%	13.5%

4.4.4 RQ3. Ablation Study

We conduct an ablation study to evaluate how different components of the LLM-based pipeline in AEAS affect feature extraction performance. Specifically, we examine the influence of (1) LLM backbones and (2) LLM techniques such as chain-of-thought (CoT) reasoning. This analysis assesses the robustness of the pipeline design and its adaptability to different model choices.

Given the high manual effort required to label ground truth at the sub-feature level, we focus on a sample of 10 randomly selected vulnerabilities, which collectively associated with 51 candidate exploits. For each exploit, we manually label six sub-features under the attack complexity feature (defined in Table 4.1), resulting in 306 labeled data points. The feature extraction process is repeated three times for each exploit and each model to measure accuracy and consistency. This setup allows us to perform feature-level evaluation while maintaining manageable annotation overhead.

LLM Backbones. We compare seven different LLM backbones on their ability to extract structured features from exploit artifacts. As shown in Table 4.7, DeepSeek-R1 achieves the highest average accuracy (83.3%), followed by o3-mini (78.7%) and DeepSeek-V3 (76.0%). Even lightweight models such as GPT-4o-mini (64.3%) and Kimi variants (62.6–67.1%) demonstrate competitive performance, highlighting the adaptability of our pipeline to different LLM backbones.

It is worth noting that our previous evaluations at the exploit and vulnerability levels were conducted using GPT-4o-mini, selected for its cost efficiency. Despite its relatively modest 64.3% feature-level accuracy, AEAS achieved strong performance in those evaluations (Section 4.4.2 and 4.4.3), demonstrating that the system is robust to moderate levels of prediction noise. As long as the LLM outputs are broadly reasonable, our aggregation and ranking procedures yield actionable results. While stronger reasoning models such as DeepSeek or o3-mini can further improve feature-level accuracy, which could potentially benefit expert users who wish to inspect specific feature attributions, lighter models already offer sufficient fidelity for general use cases such as vulnerability triage or automated red teaming.

LLM Techniques. We also examine how prompting strategies influence feature extraction by comparing performance with and without CoT reasoning. The evaluation is conducted on two OpenAI models: GPT-4o-mini, a lightweighted model without dedicated reasoning support, and

o3-mini, a more advanced variant designed to handle reasoning tasks. In both cases, we use task-specific prompts tailored to the exploit assessment. The only difference is that the CoT version includes an explicit step-by-step guide to help the model structure its reasoning. We aim to evaluate whether adding such detailed guidance improves accuracy and stability.

As shown in Table 4.8, CoT improves accuracy for both models. GPT-4o-mini improves from 59.4% to 64.3%, and o3-mini improves from 73.3% to 78.7%. However, CoT has a differential effect on variance. While it reduces variance for GPT-4o-mini (from 22.6% to 19.6%), it slightly increases variance for o3-mini (from 13.5% to 14.7%). This suggests that while CoT helps both models arrive at more accurate predictions, its stabilizing effect is more pronounced for models without built-in reasoning capabilities. In contrast, reasoning-capable models may already follow internal reasoning paths, and externally injected reasoning steps may sometimes introduce misalignment or unnecessary verbosity, leading to higher variability across runs.

Nonetheless, these results demonstrate that CoT is effective even for models designed with reasoning capabilities. This suggests that while such models may be capable of reasoning in general, they are not sufficiently trained for the specific task of exploit assessment. Our task-specific CoT prompts help compensate for this limitation by guiding the model through domain-relevant steps.

4.5 Limitation and Future Work

4.5.1 Static Analysis Without Execution

AEAS is designed to statically assess the actionability of exploits without executing them. This choice is intentional. Dynamic execution is costly, difficult to scale, and unsuitable for environments that require stealth or reproducibility. Our framework instead aims to approximate the effectiveness of runtime testing through careful extraction and interpretation of structured features. However, static analysis may miss behaviors that only emerge during execution, such as

environment-specific interactions or timing-based logic. A promising direction for future work is to integrate selective dynamic validation for ambiguous or high-risk cases, while retaining static analysis as the core strategy for scalability.

4.5.2 Dependence on Exploit Availability

AEAS requires access to exploit artifacts to perform its analysis. Although the system can process both public and private exploit repositories, it cannot evaluate vulnerabilities without available exploit code or accompanying documentation. This means the framework complements, but does not replace, predictive systems that infer exploitability in the absence of known artifacts. Future extensions could explore hybrid designs that combine artifact-driven analysis with predictive modeling to handle vulnerabilities with limited public information.

4.5.3 Impact of LLM Evolution

As large language models continue to evolve, their changing capabilities may influence the performance of AEAS. While stronger models are likely to improve feature extraction accuracy, shifts in model behavior, such as reasoning style or prompt sensitivity, could also introduce unexpected variance or inconsistencies. Our evaluation demonstrates that AEAS performs well across different model backbones, suggesting a degree of robustness. However, to ensure long-term reliability, future work could explore adaptive techniques such as prompt tuning and model calibration. Monitoring and benchmarking future LLM versions will also be important to detect performance drift and maintain consistent analysis quality.

4.5.4 Scope of Actionability Features

This work focuses specifically on three dimensions of exploit actionability: availability, functionality, and setup complexity. These dimensions reflect common bottlenecks in exploit deployment and are critical for tasks such as exploit triage and red teaming. However, they do not capture other aspects such as stealth and persistence. These properties are often scenario-dependent and harder to infer from static artifacts alone. Expanding the framework to support broader exploit attributes would require new feature designs, validation criteria, and domain-specific modeling, which we consider promising directions for future exploration.

4.6 Conclusion

In this chapter, we presented AEAS, an automated system for actionable exploit assessment. Unlike existing scoring schemes or penetration testing tools that focus on theoretical severity or artifact collection, AEAS prioritizes exploit usability by analyzing whether exploits are available, functional, and require minimal setup. Our system integrates structured feature extraction with task-specific LLM prompting and heuristic scoring to evaluate exploit quality without dynamic execution.

We evaluated AEAS using a large dataset derived from over 5,000 vulnerabilities associated with 655 real-world applications commonly encountered in red teaming operations. Through a combination of manual validation and expert feedback, we demonstrated that AEAS can effectively identify usable exploits and produce vulnerability rankings that align with expert expectations. In particular, the system achieved a 100% success rate in its top-3 exploit recommendations on a manually verified subset, highlighting its practical value in automated red teaming and vulnerability triage workflows.

4.A Appendix

4.A.1 Pre-processing Details

We utilize publicly available information on Google and GitHub as primary data sources due to their broad coverage of relevant information. However, the search results will often contain a lot of irrelevant data, making it necessary to pre-process the search results to eliminate irrelevant data while retaining meaningful and high-quality information. This ensures that subsequent stages are based on reliable inputs, enhancing the overall efficacy of the system.

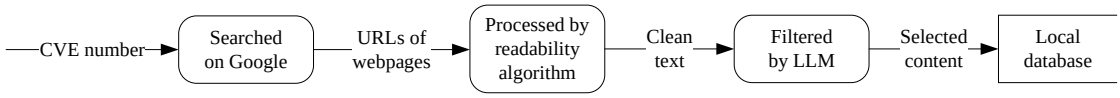


Figure 4.7: Process of Collecting Data on Google

Google serves as a powerful search engine, providing access to a vast array of web content. However, the variability in search result quality requires a systematic approach to processing and filtering the retrieved data. We begin by extracting webpage content from Google search results and converting HTML documents into text format. Media files, such as images, videos, and audio files, are excluded, as our focus is on analyzing textual data.

To ensure relevance, we apply content-based filtering using the readability algorithm [121], [122] to remove irrelevant sections of webpages, such as sidebars and advertisements, while keeping critical content like code snippets and documentation as shown in Fig. 4.7. Additionally, we utilize an LLM agent with carefully designed prompts incorporating role-playing and few-shot learning techniques. The LLM agent is guided to filter out irrelevant content and emphasize actionable, exploitation-relevant information, such as executable code snippets. We specifically exclude summarized CVE data from vulnerability databases, as such summaries often lack the technical depth necessary for exploitation analysis.

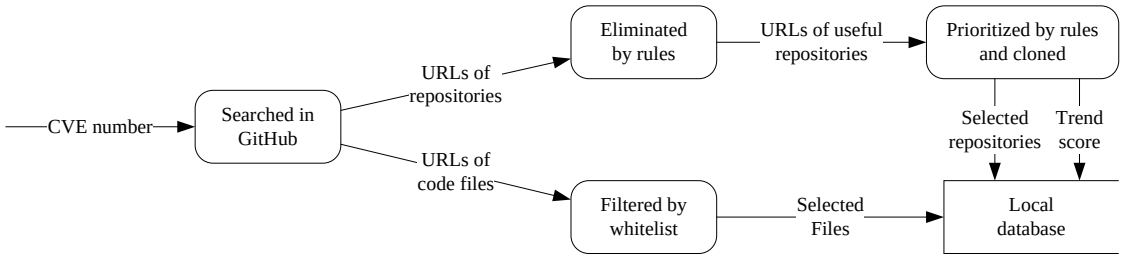


Figure 4.8: Process of Collecting Data on GitHub

GitHub offers a wealth of repositories containing ready-to-use code, making it a valuable resource for vulnerability assessments. However, the variability in repository quality necessitates a robust filtering and prioritization framework. Our approach incorporates rule-based processing at both the repository and file levels to ensure the relevance and quality of the dataset as outlined in Fig. 4.8.

Table 4.9: Heuristic Rule Set for Filtering Repositories

Task	Criteria
Elimination	Excessively long descriptions Abnormal issue counts Large number of irrelevant tags
Prioritization	Too large or too small sizes High numbers of forks Low numbers of stars

At the repository level, we implement a two-step process: (1) *Elimination* to exclude low-quality or irrelevant repositories and (2) *Prioritization* to rank higher-quality repositories when multiple candidates are available. Repository quality is assessed using heuristic rules based on metadata attributes, as we outlined in Table 4.9. For Elimination, a confidence score is calculated for each repository:

$$\textit{Confidence Score} = w_1 \times d + w_2 \times i + w_3 \times t \tag{6}$$

where d represents the description length, i accounts for issue counts, and t reflects topics associated with the repository. Repositories scoring below a predefined threshold are excluded. For Prioritization, repositories are ranked using the quality score:

$$\textit{Quality Score} = \frac{\textit{Stars} \cdot \lambda \cdot \Delta T}{\textit{Forks}} \tag{7}$$

where ΔT represents the age of the repository, and λ is a normalization parameter to account for differences in creation time. Repositories with higher stars and fewer forks are preferred, as heavily forked repositories often indicate experimental or incomplete scripts unsuitable for direct use.

At the file level, we focus on keeping files that provide meaningful information for vulnerability assessment. These include source code written in high-level programming languages and README files containing detailed descriptions and usage instructions. Conversely, we remove binary files, media files, and configuration files (e.g., XML) that typically lack the content needed for effective analysis. This targeted approach ensures that the kept files are both relevant and suitable for downstream processing.

By meticulously applying these rule-based and LLM-driven filtering and scoring mechanisms, we ensure that the data is not only relevant but also optimized for subsequent analysis.

4.A.2 AEAS Output Example

The following examples show outputs of sub-feature extractions. The results are presented in a structured format, including the file reference, the extracted information, and the conclusion drawn from the analysis with confidence levels.

Information Dependency Analysis

File Analysis

1. CVE-2023-42793.py:47 - Hardcoded credentials (Technique ID 1) - Uses hardcoded password "@H454NSec" for created admin accounts
2. CVE-2023-42793.py:32 - Dynamic credential acquisition (Technique ID 2) - Generates random usernames but uses fixed password

Conclusion

True

Confidence

5

Uncontrollable Preconditions Analysis

File Analysis

1. CVE-2023-42793.py:75 - Hardcoded dependencies (Technique ID 1) - Requires specific TeamCity API endpoints (/app/rest/users/id:1/tokens/RPC2)
2. CVE-2023-42793.py:1 - Third-party service API requirements (Technique ID 2) - Requires vulnerable TeamCity REST API to be accessible

Conclusion

True

Confidence

5

Probabilistic Analysis

File Analysis

1. CVE-2023-42793.py:15 - Randomness-dependent execution paths (Technique ID 3) - Uses random username generation (1000-9999 range) but this doesn't affect exploit success

Conclusion

False

Confidence

4

4.A.3 Additional Evaluation Details

To complement the Bland-Altman analysis, we calculated additional metrics to quantify the alignment between AEAS and EPSS scores. Table 4.10 summarizes these metrics, including Pearson Correlation, Spearman Correlation, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). For the full dataset, the metrics indicate moderate to strong correlations (Pearson: 0.5772, Spearman: 0.6293) and low error rates (MAE: 0.1979, RMSE: 0.3087). For vulnerabilities with associated exploits, the metrics show improved precision, with lower MAE (0.1208) and RMSE (0.1788), despite a reduced correlation values. Intuitively, vulnerabilities without available exploits are generally harder to exploit, resulting in lower default scores in AEAS that align with EPSS's generally lower scores, thereby inflating correlation values.

Table 4.10: Vulnerability Severity Agreement Analysis

Metric	**Value (All)**	**Value (w/ Exploits)**
Pearson Correlation	0.5772	0.4439
Spearman Correlation	0.6293	0.4924
MAE	0.1979	0.1208
RMSE	0.3087	0.1788

In summary, AEAS demonstrates a reasonable agreement with EPSS scores, suggesting that it can effectively assess vulnerability severity for most cases. We also performed an outlier analysis to identify vulnerabilities with significant discrepancies between AEAS and EPSS scores, which is detailed in Section 4.4.3 of the main text.

CHAPTER 5
CONCLUSION AND FUTURE WORK

This dissertation began with a practical challenge: while security vendors and defenders increasingly rely on adversary emulation to test detection systems, existing evaluation efforts are difficult to interpret and limited in scope. The MITRE ATT&CK Evaluations, while influential, expose only a narrow slice of defensive performance and leave critical questions unanswered, such as whether detection is timely, correlated across attack stages, or complete in the context of a full kill chain. This motivated the first part of the dissertation, which introduced Decoding MITRE ATT&CK Evaluations, a graph-based analysis framework for systematically interpreting adversary emulation results. By introducing structured metrics and modeling relationships between detections, this work transformed coarse, static evaluation outputs into actionable insights for defenders, researchers, and vendors alike.

However, this analysis also revealed a deeper problem: defense evaluation efforts are fundamentally constrained by the limited availability of high-quality, flexible attack data. To understand and improve detection systems continuously, organizations require more than post hoc analysis of static datasets. They need support for scalable, realistic, and explainable adversary simulations. This realization led to the second thrust of the dissertation: building systems to facilitate the red teaming process itself.

To this end, I developed PentestAgent, a multi-agent, LLM-driven penetration testing framework that automates key phases of red team operations. By decomposing attack workflows into specialized agents, such as reconnaissance, planning, and exploitation, and enabling them to reason over goals and environment feedback, PentestAgent supports realistic attack emulation with

minimal manual effort. It enables more scalable, consistent, and reproducible red teaming that can be integrated into continuous security validation pipelines.

Even with scalable execution, the planning phase remains critical. Poor exploit choices can undermine the effectiveness of an entire red team engagement. This led to the third contribution: AEAS (Actionable Exploit Assessment System), a framework for evaluating public exploit artifacts in terms of availability, functionality, and setup complexity. AEAS combines large language model reasoning with structured feature extraction to produce actionability scores, severity assessments, and natural-language justifications. By enabling both human and automated red teamers to prioritize high-quality, operationally ready exploits, AEAS improves planning efficiency and effectiveness.

Together, these three systems form a cohesive response to the broader challenge of making red teaming more explainable, scalable, and operationally relevant. Rather than attempting to build attack techniques from scratch, this dissertation focuses on facilitating and improving the processes that underlie modern adversary simulation and evaluation: linking evaluation with execution, automation with explanation, and offensive realism with defensive insight. This integrated approach supports a vision of proactive security testing where defenders can continuously assess and improve their systems in the face of evolving threats.

5.1 Reflections and Lessons Learned

Several key themes emerged over the course of this research. First, structure matters. Whether evaluating detection pipelines or selecting exploits, systems that apply structured representations, such as graphs, taxonomies, or agent roles, enable greater transparency, reproducibility, and adaptability. Second, language models offer powerful reasoning capabilities, but their effectiveness is significantly enhanced when paired with retrieval, context constraints, and task-specific decom-

position. Finally, offensive and defensive workflows should not be developed in isolation. Each can inform and improve the other, and building tools that expose this interplay results in more meaningful and actionable outcomes for practitioners.

5.2 Opportunities for future research

The systems and techniques presented in this dissertation open up several promising directions for future research:

- **Enhanced integration of offensive and defensive workflows:** Future work could explore deeper integration between the systems, such as using insights from Decoding MITRE to inform PentestAgent's attack simulations or to refine detection capabilities.
- **Context-aware vulnerability prioritization:** Building on AEAS, future research could develop context-aware prioritization frameworks that consider not only exploit actionability but also the specific characteristics of an organization's environment, such as network topology, asset criticality, and existing defenses.
- **Multimodal analysis for vulnerability assessment:** Expanding AEAS to incorporate multimodal data sources, such as images, videos, and audio, could enhance its ability to analyze complex exploits and vulnerabilities. This would require developing new models capable of processing diverse data types while maintaining explainability.
- **Ethical considerations in automated security:** As automation becomes more prevalent in security workflows, future research should also address the ethical implications of these systems. This includes ensuring that automated tools do not inadvertently introduce biases, respect user privacy, and operate transparently to maintain trust among stakeholders.

REFERENCES

[1] B. Barrett, *How 4 Chinese Hackers Allegedly Took Down Equifax*, https://www.wired.com/story/equifax-hack-china/, 2020.

[2] CNNMoney, *Target: 40 million credit cards compromised*, https://money.cnn.com/2013/12/18/news/companies/target-credit-card/index.html, Dec. 2013.

[3] W. J. Broad, J. Markoff, and D. E. Sanger, *Israeli Test on Worm Called Crucial in Iran Nuclear Delay*, https://www.nytimes.com/2011/01/16/world/middleeast/16stuxnet.html, Jan. 2011.

[4] CompaniesMarketCap, *Largest companies by market cap*, https://companiesmarketcap.com/it-security/largest-companies-by-market-cap/, 2021.

[5] "Endpoint detection and response market report." ().

[6] Gartner, *Magic Quadrant for Endpoint Protection Platforms*, https://www.gartner.com/en/documents/4001307/magic-quadrant-for-endpoint-protection-platforms, 2021.

[7] AV-Comparatives, *AV-Comparatives*, https://www.av-comparatives.org/, 2023.

[8] DARPA, *DARPA Intrusion Detection Evaluation*, 1998.

[9] R. Zuech, T. M. Khoshgoftaar, N. Seliya, M. M. Najafabadi, and C. Kemp, "A new intrusion detection benchmarking system," in *Florida Artificial Intelligence Research Society Conference*, 2015.

[10] A. Divekar, M. Parekh, V. Savla, R. Mishra, and M. Shirole, "Benchmarking datasets for anomaly-based network intrusion detection: Kdd cup 99 alternatives," in *IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, 2018.

[11] N. S. Almakhdhub, A. A. Clements, M. Payer, and S. Bagchi, "Benchiot: A security benchmark for the internet of things," in *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[12] DARPA, *DARPA Transparent Computing*, 2023.

[13] G. Hao, F. Li, W. Huo, *et al.*, "Constructing benchmarks for supporting explainable evaluations of static application security testing tools," in *International Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2019.

[14] N. Mendes, H. Madeira, and J. Duraes, "Security benchmarks for web serving systems," in *IEEE 25th International Symposium on Software Reliability Engineering*, 2014.

[15] MITRE, *ATT&CK® EVALUATIONS*, `https://attackevals.mitre-engenuity.org/`, 2023.

[16] MITRE, *Matrix - Enterprise — MITRE ATT&CK®*, `https://attack.mitre.org/matrices/enterprise/`, 2023.

[17] A. Bates, D. (Tian, K. R. Butler, and T. Moyer, "Trustworthy Whole-System provenance for the linux kernel," in *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C.: USENIX Association, Aug. 2015, pp. 319–334, ISBN: 978-1-939133-11-3.

[18] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03, Bolton Landing, NY, USA: Association for Computing Machinery, 2003, pp. 223–236, ISBN: 1581137575.

[19] Crowdstrike, *CrowdStrike Achieves 100% Detection Coverage*, `https://www.crowdstrike.com/blog/crowdstrike-falcon-mitre-attack-evaluation-results-third-iteration/`, 2021.

[20] C. for Threat-Informed Defense, *Adversary emulation library*, `https://github.com/center-for-threat-informed-defense/adversary_emulation_library`, Accessed: 2023.

[21] CrowdStrike. "Kerberoasting - cybersecurity 101." (2023).

[22] 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.

[23] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," *ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[24] 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.

[25] C. Xiong, T. Zhu, W. Dong, *et al.*, "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, Feb. 2020.

[26] M. N. Hossain, S. M. Milajerdi, J. Wang, *et al.*, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *USENIX Security Symposium*, Aug. 2017, ISBN: 978-1-931971-40-9.

[27] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *Network and Distributed System Security Symposium*, 2020.

[28] W. U. Hassan, S. Guo, D. Li, *et al.*, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed System Security Symposium*, 2019.

[29] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting," in *Network and Distributed System Security Symposium*, 2016.

[30] Q. Wang, W. U. Hassan, D. Li, *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Network and Distributed System Security Symposium*, 2020.

[31] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2020.

[32] Y. Xie, Y. Wu, D. Feng, and D. Long, "P-gaussian: Provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2658–2674, 2021.

[33] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics.," in *NDSS*, 2021.

[34] J. Zengy, X. Wang, J. Liu, *et al.*, "Shadewatcher: Recommendation-guided cyber threat analysis using system audit records," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2022, pp. 489–506.

[35] A. Alsaheel, Y. Nan, S. Ma, *et al.*, "Atlas: A sequence-based learning approach for attack investigation.," in *USENIX Security Symposium*, 2021, pp. 3005–3022.

[36] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: Scalable and accurate zero-day android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services*, 2012, ISBN: 9781450313018.

[37] A. K. Marnerides, P. Spachos, P. Chatzimisios, and A. U. Mauthe, "Malware detection in the cloud under ensemble empirical mode decomposition," in *International Conference on Computing, Networking and Communications (ICNC)*, 2015.

[38] M. Zipperle, F. Gottwalt, E. Chang, and T. Dillon, "Provenance-based intrusion detection systems: A survey," *ACM Comput. Surv.*, vol. 55, no. 7, Dec. 2022.

[39] Z. Li, Q. A. Chen, R. Yang, Y. Chen, and W. Ruan, "Threat detection and investigation with system-level provenance graphs: A survey," *Computers & Security*, vol. 106, p. 102 282, 2021.

[40] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, 2019.

[41] E. van der Kouwe, G. Heiser, D. Andriesse, H. Bos, and C. Giuffrida, "Sok: Benchmarking flaws in systems security," in *IEEE European Symposium on Security and Privacy (EuroS P)*, 2019.

[42] 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*, ser. CSET '21, Virtual, CA, USA: Association for Computing Machinery, 2021, pp. 41–48, ISBN: 9781450390651.

[43] A. V. Outkin, P. V. Schulz, T. Schulz, T. D. Tarman, and A. Pinar, "Defender policy evaluation and resource allocation with mitre att&ck evaluations data," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 3, pp. 1909–1926, 2023.

[44] B. A. Alahmadi, L. Axon, and I. Martinovic, "99% false positives: A qualitative study of SOC analysts' perspectives on security alarms," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 2783–2800, ISBN: 978-1-939133-31-1.

[45] G. P. Spathoulas and S. K. Katsikas, "Using a fuzzy inference system to reduce false positives in intrusion detection," in *2009 16th International Conference on Systems, Signals and Image Processing*, 2009, pp. 1–4.

[46] M. Denis, C. Zena, and T. Hayajneh, "Penetration testing: Concepts, attack methods, and defense strategies," in *2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, IEEE, 2016, pp. 1–6.

[47] Y. Stefinko, A. Piskozub, and R. Banakh, "Manual and automated penetration testing. benefits and drawbacks. modern tendency," in *2016 13th international conference on modern problems of radio engineering, telecommunications and computer science (TCSET)*, IEEE, 2016, pp. 488–491.

[48] R. G. Consulting, *Under the hoodie: Lessons from a season of penetration testing*, Accessed: 2024-06-19, 2019.

[49] 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.

[50] J. L. Obes, C. Sarraute, and G. Richarte, "Attack planning in the real world," *arXiv preprint arXiv:1306.4044*, 2013.

[51] M. Roberts, A. Howe, I. Ray, M. Urbanska, Z. S. Byrne, and J. M. Weidert, "Personalized vulnerability analysis through automated planning," in *Working Notes for the 2011 IJCAI Workshop on Intelligent Security (SecArt)*, 2011, p. 50.

[52] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 217–224.

[53] K. Durkota and V. Lisỳ, "Computing optimal policies for attack graphs with action failures and costs.," in *STAIRS*, 2014, pp. 101–110.

[54] L. Krautsevich, F. Martinelli, and A. Yautsiukhin, "Towards modelling adaptive attacker's behaviour," in *Foundations and Practice of Security: 5th International Symposium, FPS 2012, Montreal, QC, Canada, October 25-26, 2012, Revised Selected Papers 5*, Springer, 2013, pp. 357–364.

[55] C. Sarraute, G. Richarte, and J. Lucángeli Obes, "An algorithm to find optimal attack paths in nondeterministic scenarios," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, 2011, pp. 71–80.

[56] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing== pomdp solving?" *arXiv preprint arXiv:1306.4714*, 2013.

[57] C. Sarraute, O. Buffet, and J. Hoffmann, "Pomdps make better hackers: Accounting for uncertainty in penetration testing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, 2012, pp. 1816–1824.

[58] T.-y. Zhou, Y.-c. Zang, J.-h. Zhu, and Q.-x. Wang, "Nig-ap: A new method for automated penetration testing," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, no. 9, pp. 1277–1288, 2019.

[59] 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.

[60] 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. 103 055, 2023.

[61] G. Deng, Y. Liu, V. Mayoral-Vilches, *et al.*, "Pentestgpt: Evaluating and harnessing large language models for automated penetration testing," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 847–864.

[62] 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.

[63] J. Xu, J. W. Stokes, G. McDonald, *et al.*, "Autoattacker: A large language model guided system to implement automatic cyber-attacks," *arXiv preprint arXiv:2403.01038*, 2024.

[64] P. Lewis, E. Perez, A. Piktus, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[65] T. P. T. E. Standard, *Ptes technical guidelines*, 2024.

[66] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large language model guided protocol fuzzing," in *Proceedings of the 31st Annual Network and Distributed System Security Symposium (NDSS)*, 2024.

[67] Y. Deng, C. S. Xia, C. Yang, S. D. Zhang, S. Yang, and L. Zhang, "Large language models are edge-case generators: Crafting unusual programs for fuzzing deep learning libraries," in *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*, 2024, pp. 1–13.

[68] H. Li, Y. Hao, Y. Zhai, and Z. Qian, "The hitchhiker's guide to program analysis: A journey with large language models," *arXiv preprint arXiv:2308.00245*, 2023.

[69] P. Liu, C. Sun, Y. Zheng, *et al.*, "Harnessing the power of llm to support binary taint analysis," *arXiv preprint arXiv:2310.08275*, 2023.

[70] R. G. Consulting, *Under the hoodie: Lessons from a season of penetration testing*, Accessed: 2024-06-27, 2020.

[71] nmap, *Nmap*, 2024.

[72] Tenable, *Tenable nessus*, 2024.

[73] GreenBone, *Greenbone openvas*, 2024.

[74] Rapid7, *Rapid7 metasploit*, 2024.

[75] Microsoft, *System message framework and template recommendations for large language models (llms)*, 2024.

[76] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[77] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "Camel: Communicative agents for" mind" exploration of large language model society," *Advances in Neural Information Processing Systems*, vol. 36, pp. 51 991–52 008, 2023.

[78] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[79] M. X. Liu, F. Liu, A. J. Fiannaca, *et al.*, "" we need structured output": Towards user-centered constraints on large language model output," in *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–9.

[80] G. Deng, Y. Liu, Y. Li, *et al.*, "Jailbreaker: Automated jailbreak across multiple large language model chatbots," *arXiv preprint arXiv:2307.08715*, 2023.

[81] 0x727, *Observerward*, 2024.

[82] S. Security, *Snyk vulnerability database*, 2024.

[83] A. Could, *Vulnerability db*, 2024.

[84] HackTheBox, *Hackthebox: Hacking training for the best.* 2024.

[85] OWASP, *Owasp benchmark*, 2024.

[86] VulnHub, *Vulnhub*, 2024.

[87] Vulhub, *Vulhub*, 2024.

[88] MITRE, *Cve*, 2024.

[89] N. V. Database, *Common vulnerability scoring system calculator*, 2024.

[90] F. of Incident Response and I. Security Teams, *Exploit prediction scoring system (epss)*, 2024.

[91] OWASP, *Top 10 web application security risks*, 2024.

[92] AutoGPT, *Autogpt*, 2024.

[93] T. Abramovich, M. Udeshi, M. Shao, *et al.*, "Enigma: Enhanced interactive generative model agent for ctf challenges," *arXiv preprint arXiv:2409.16165*, 2024.

[94] F. of Incident Response and I. Security Teams, *Common vulnerability scoring system v3.0: Specification document*, 2024.

[95] J. Wunder, A. Kurtz, C. Eichenmüller, F. Gassmann, and Z. Benenson, "Shedding light on cvss scoring inconsistencies: A user-centric study on evaluating widespread security vulnerabilities," in *2024 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2024, pp. 1102–1121.

[96] S. Zhang, M. Cai, M. Zhang, L. Zhao, and X. d. C. de Carnavalet, "The flaw within: Identifying cvss score discrepancies in the nvd," in *2023 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2023, pp. 185–192.

[97] L. Allodi and F. Massacci, "Comparing vulnerability severity and exploits using case-control studies," *ACM Transactions on Information and System Security (TISSEC)*, vol. 17, no. 1, pp. 1–20, 2014.

[98] J. Jacobs, S. Romanosky, O. Suciu, B. Edwards, and A. Sarabi, "Enhancing vulnerability prioritization: Data-driven exploit predictions with community-driven insights," in *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2023, pp. 194–206.

[99] Tenable, *Tenable vulnerability management*, 2024.

[100] P. Discovery, *Cvemap*, 2024.

[101] P. Discovery, *Nuclei*, 2024.

[102] V. Ashiwal, S. Finster, and A. Dawoud, "Llm-based vulnerability sourcing from unstructured data," in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2024, pp. 634–641.

[103] K. Milousi, P. Kiriakidis, N. Mengidis, *et al.*, "Evaluating cybersecurity risk: A comprehensive comparison of vulnerability scoring methodologies," in *Proceedings of the 19th International Conference on Availability, Reliability and Security*, 2024, pp. 1–11.

[104] J. Jacobs, S. Romanosky, I. Adjerid, and W. Baker, "Improving vulnerability remediation through better exploit prediction," *Journal of Cybersecurity*, vol. 6, no. 1, tyaa015, 2020.

[105] T. H. Le, H. Chen, and M. A. Babar, "A survey on data-driven software vulnerability assessment and prioritization," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–39, 2022.

[106] J. Yin, G. Chen, W. Hong, H. Wang, J. Cao, and Y. Miao, "Empowering vulnerability prioritization: A heterogeneous graph-driven framework for exploitability prediction," in *International Conference on Web Information Systems Engineering*, Springer, 2023, pp. 289–299.

[107] Y. Fang, Y. Liu, C. Huang, and L. Liu, "Fastembed: Predicting vulnerability exploitation possibility based on ensemble machine learning algorithm," *Plos one*, vol. 15, no. 2, e0228439, 2020.

[108] N. S. Harzevili, A. B. Belle, J. Wang, S. Wang, Z. Ming, N. Nagappan, *et al.*, "A survey on automated software vulnerability detection using machine learning and deep learning," *arXiv preprint arXiv:2306.11673*, 2023.

[109] O. Suciu, C. Nelson, Z. Lyu, T. Bao, and T. Dumitraș, "Expected exploitability: Predicting the development of functional vulnerability exploits," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 377–394.

[110] R. 7, *Metasploit documentation - exploit ranking*, 2024.

[111] E. Fedorchenko, E. Novikova, A. Fedorchenko, and S. Verevkin, "An analytical review of the source code models for exploit analysis," *Information*, vol. 14, no. 9, p. 497, 2023.

[112] H. Li, H. Gao, C. Wu, and M. A. Vasarhelyi, "Extracting financial data from unstructured sources: Leveraging large language models," *Journal of Information Systems*, pp. 1–22, 2023.

[113] I. C. Wiest, F. Wolf, M.-E. Leßmann, *et al.*, "Llm-aix: An open source pipeline for information extraction from unstructured medical text based on privacy preserving large language models," *medRxiv*, 2024.

[114] W. Wang, Z. Ma, Z. Wang, *et al.*, "A survey of llm-based agents in medicine: How far are we from baymax?" *arXiv preprint arXiv:2502.11211*, 2025.

[115] T. Lin, W. Zhang, S. Li, *et al.*, "Healthgpt: A medical large vision-language model for unifying comprehension and generation via heterogeneous knowledge adaptation," *arXiv preprint arXiv:2502.09838*, 2025.

[116] R. Fayyazi and S. J. Yang, "On the uses of large language models to interpret ambiguous cyberattack descriptions," *arXiv preprint arXiv:2306.14062*, 2023.

[117] M. D. Purba, A. Ghosh, B. J. Radford, and B. Chu, "Software vulnerability detection using large language models," in *2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW)*, IEEE, 2023, pp. 112–119.

[118] R. I. T. Jensen, V. Tawosi, and S. Alamir, "Software vulnerability and functionality assessment using llms," in *2024 IEEE/ACM International Workshop on Natural Language-Based Software Engineering (NLBSE)*, IEEE, 2024, pp. 25–28.

[119] X. Zhou, T. Zhang, and D. Lo, "Large language model for vulnerability detection: Emerging results and future directions," in *Proceedings of the 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*, 2024, pp. 47–51.

[120] T. Ahmed and P. Devanbu, "Better patching using llm prompting, via self-consistency," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2023, pp. 1742–1746.

[121] K. Collins-Thompson and J. P. Callan, "A language modeling approach to predicting reading difficulty," in *Proceedings of the human language technology conference of the North American chapter of the association for computational linguistics: HLT-NAACL*, 2004, pp. 193–200.

[122] S. E. Schwarm and M. Ostendorf, "Reading level assessment using support vector machines and statistical language models," in *Proceedings of the 43rd annual meeting of the Association for Computational Linguistics (ACL'05)*, 2005, pp. 523–530.

BEYOND EVALUATION: TOWARDS AUTOMATED AND EXPLAINABLE RED TEAMING

Approved by:

Yan Chen
Computer Science
Northwestern University

Xinyu Xing
Computer Science
Northwestern University

Han Liu
Computer Science
Northwestern University

Zhenkai Liang
School of Computing
National University of Singapore

Date Approved: July 8, 2025