

End-to-End Attack Scene Reconstruction in a Host With Rules and Anomaly-Based Detection Models

Su Wang¹, Hongbin Sun¹, Zhiliang Wang¹, *Member, IEEE*, Tao Zhou, Xia Yin, *Senior Member, IEEE*, Dongqi Han, Han Zhang², Xingang Shi², *Member, IEEE*, and Jiahai Yang², *Senior Member, IEEE*

Abstract—Critical devices on the Internet are frequently targeted by skilled and advanced network attackers. These attackers often orchestrate complex and persistent intrusion campaigns, which involve multiple stages of attacks. In the context of host-based threat detection, the reconstruction of the entire attack scenario is crucial for tracing threats and fixing system vulnerabilities. Prior anomaly-based studies lack the capability to interpret the attack scenario, while rule-based approaches struggle with detecting novel attack patterns. We introduce EAGLE, an end-to-end framework that takes original host-based data as input and reconstructs the potential attack scenario as output. It leverages an anomaly-based algorithm and a fine-grained misuse detection module to assign anomalous scores to host data, constructs the potential attack scenario using a novel anomalous subtree detection algorithm, and generates the interpretable attack scenario graph through a coarse-grained rule matching method. We assess the performance of EAGLE using three attack scenarios from the DARPA TC dataset and three deployment scenarios. The results demonstrate that EAGLE can effectively uncover the hidden attack scenario within the host data and outperforms three state-of-the-art attack scenario reconstruction systems.

Index Terms—Host-based attack scene reconstruction, graph neural network, data provenance, rule matching, anomaly-based algorithm.

I. INTRODUCTION

ADVANCED threats are frequently crafted and deployed by network attackers to infiltrate crucial hosts within companies and government departments. These sophisticated threats tend to persist for long periods, concealing their campaigns within systems and posing significant risks to

important assets [1], [2]. Intrusion campaigns of this nature are embedded within vast amounts of host data, making detection challenging. The frequent exploitation of zero-day vulnerabilities adds another layer of challenge to detection systems.

We study the intrusion campaigns occurring on the host. In the realm of host-based intrusion detection, detecting and reconstructing the complete attack scene has become a research focus. To effectively detect advanced threats on the host, selecting appropriate data sources poses an initial issue. An ideal data source should contain rich information for intrusion analysis. The provenance graph is proposed as a suitable data source for host-based intrusion detection [3], [4], [5], [6], [7]. It is a directed acyclic graph constructed from system audit data, where nodes represent system entities such as processes and files, and edges represent system calls between entities, indicating the flow of information. The provenance graph contains rich contextual information, making it potent for detecting long-term threats. Moreover, the information about causal relationships stored in the provenance graph facilitates causal analysis of different attack steps for the detection system.

Since Backtracker [8], the first work to use provenance graphs for system intrusion investigation, was proposed in 2003, an increasing number of studies on threat detection and reconstruction based on provenance graphs have been introduced [9]. They can generally be categorized into three categories: anomaly-based, rule-based, and hybrid methods. Anomaly-based methods [3], [4], [5], [10], [11] create models for benign activities and detect threats based on deviations from these models. Although these approaches have the potential to hunt for novel threats, they often lack the ability to provide detailed explanations for the alerts, making it difficult to interpret the attack scenario [4]. To interpret the attack scenarios, they need to use external interpretation tools [12], [13], [14] and thus rely on the performance of these tools. Interpretability is crucial for scene reconstruction algorithms because the stronger the interpretability, the lower the required manual labor costs [3], [15]. Rule-based approaches [6], [7], [16] design attack patterns based on expert knowledge and prior experience. These attack patterns, often referred to as rulesets, enable the detection of threats through pattern matching between the provenance graph and the ruleset. Unfortunately, due to the limitations of predefined rules, rule-based approaches often struggle to detect zero-day attacks. Both of these detection approaches possess distinctive

Received 10 October 2024; revised 30 April 2025 and 2 July 2025; accepted 4 July 2025. Date of publication 11 July 2025; date of current version 18 July 2025. This work was supported by the Zhongguancun Laboratory. The associate editor coordinating the review of this article and approving it for publication was Dr. Fabio De Gaspari. (*Corresponding authors: Zhiliang Wang; Xia Yin.*)

Su Wang and Hongbin Sun are with the Zhongguancun Laboratory, Beijing 100084, China (e-mail: wangsu@zgclab.edu.cn; sunhb@zgclab.edu.cn).

Zhiliang Wang, Han Zhang, Xingang Shi, and Jiahai Yang are with the Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University, Beijing 100084, China, and also with the Zhongguancun Laboratory, Beijing 100084, China (e-mail: wzl@cernet.edu.cn; zhhan@tsinghua.edu.cn; shixg@cernet.edu.cn; yang@cernet.edu.cn).

Tao Zhou is with Venustech Group Inc., Beijing 100084, China (e-mail: zhoutao@venustech.com.cn).

Xia Yin is with the Department of Computer Science and Technology, BNRist, Tsinghua University, Beijing 100084, China, and also with the Zhongguancun Laboratory, Beijing 100084, China (e-mail: yxia@tsinghua.edu.cn).

Dongqi Han is with the School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100084, China (e-mail: handongqi@bupt.edu.cn).

Digital Object Identifier 10.1109/TIFS.2025.3588251

1556-6021 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.

See <https://www.ieee.org/publications/rights/index.html> for more information.

Authorized licensed use limited to: Tsinghua University. Downloaded on October 15, 2026 at 12:56:20 UTC from IEEE Xplore. Restrictions apply.

characteristics, making it worthwhile to explore how to leverage their advantages concurrently. Hybrid methods [17], [18] combine various approaches to detect intrusion behavior in cyber space security. These methods are primarily applied in network threat detection, while there is a lack of related research in the field of host-based threat scenario reconstruction.

In a nutshell, anomaly-based methods detect threats by identifying deviations from models of benign behavior, but they struggle to provide detailed interpretations for alerts. Rule-based methods utilize predefined attack patterns, which can limit their effectiveness against zero-day attacks. Hybrid methods merge different approaches for network threat detection, yet there is limited research on host-based scenario reconstruction.

We introduce EAGLE, an end-to-end framework designed for host-based threat scene detection and reconstruction. EAGLE operates as a host-based system, aiming to harness the strengths of both anomaly-based and rule-based methods while proposing a novel attack scene reconstruction algorithm. By taking the provenance graph as input, EAGLE employs both anomaly-based detection and attack pattern matching approaches to generate preliminary anomaly scores for entities within a host. We employ these two types of methods to detect as many potentially threat-related nodes as possible, while leveraging the inherent interpretability of pattern matching to enhance the interpretability of the ultimately reconstructed attack scenario. We customize a Graph Attention Networks (GAT) model [19] for the anomaly-based detection task. The GAT model is designed to uncover hidden information within benign nodes in a provenance graph and identify anomalous nodes based on detected deviations. Additionally, a set of rules based on Tactics, Techniques, and Procedures (TTPs) [20] is applied for the attack pattern matching task. To pinpoint the threat scenario, we design an anomalous subtree detection and pruning algorithm. Finally, a coarse-grained rule matching method is implemented to interpret the unknown steps of the attack scene.

EAGLE incorporates two key designs. The first design addresses bridging the gap between anomaly-based and rule-based methods. By leveraging both anomaly-based and rule-based approaches, EAGLE allows the anomaly-based detection and attack pattern matching components to be replaced by distinct methods. Consequently, an algorithm is necessary to bridge the gap between these two methods. We propose an anomalous subtree detection algorithm to achieve this objective. The provenance graph is input into the anomaly-based detection and attack pattern matching modules of EAGLE. These modules contribute to the anomaly scores for each node in the graph. Subsequently, the anomalous subtree detection algorithm identifies a node as the root node of the anomalous subtree using a novel anomaly score propagation algorithm. This algorithm bridges the gap between anomaly-based and rule-based modules, resulting in an anomalous subtree for subsequent modules to analyze. The anomalous subtree detection algorithm is instrumental in reducing false positives, as it locates the anomalous campaigns in a more specific location. The second key design involves the use of attack

patterns (ruleset) to address the challenge of interpreting attack steps. The granularity of the ruleset is a perennial research challenge in the host-based threat detection community. If the granularity is too coarse, it generates more false positives. Conversely, if the granularity is too fine, some attack steps may go undetected. Based on existing research, we define two types of attack patterns: fine-grained ruleset and coarse-grained ruleset. We utilize the fine-grained ruleset to generate the anomaly score for the attack pattern matching module and employ the coarse-grained ruleset to interpret the attack steps detected by the anomaly-based detection module. These two kinds of rulesets are both generated from the TTPs [20].

Our paper makes contributions summarized as follows:

- **General framework for host-based threats scene reconstruction.** We propose a general framework that effectively combines both anomaly-based and rule-based methods. We devise an anomaly-based algorithm, subtree detection algorithm, and ruleset design strategy, enabling the accurate reconstruction of potential attack scenes within a host. We evaluate the framework with various anomaly-based detection methods to demonstrate the generality.
- **Better performance.** We conduct an extensive evaluation of EAGLE's detection capabilities using three public datasets and three simulated engagements. In comparison with three state-of-the-art host-based attack scene reconstruction methods, our results demonstrate that EAGLE excels in reconstructing interpretable attack scenes and outperforms previous approaches.
- **Open-source system.** We implement a prototype system and make the code open-source.¹

This paper is organized as follows: We introduce the background and motivation of our work in §II. The threat model is introduced in §III. The overall design and workflow of EAGLE are presented in §IV. The experiments are detailed in §V. We discuss various issues and limitations in §VI. The related work is introduced in §VII, and we conclude this paper in §VIII.

II. BACKGROUND & MOTIVATION

In this section, we introduce the concept of provenance graph, which is the data source of the attack reconstruction framework. We introduce the motivation behind our work using a toy example.

A. Provenance Graph

In this paper, we utilize the provenance graph as our primary data source. The end-to-end framework of EAGLE takes the provenance graph as input and produces the reconstructed attack scene. The provenance graph is proposed as a highly effective data source for host-based threat detection and scene reconstruction. This graph is constructed from system audit data using specific provenance generation tools [21], [22], [23].

The resulting graph is a directed acyclic graph, denoted as G . Throughout the paper, we use the same notations. V

¹<https://github.com/winsen-alpha/eaGle>

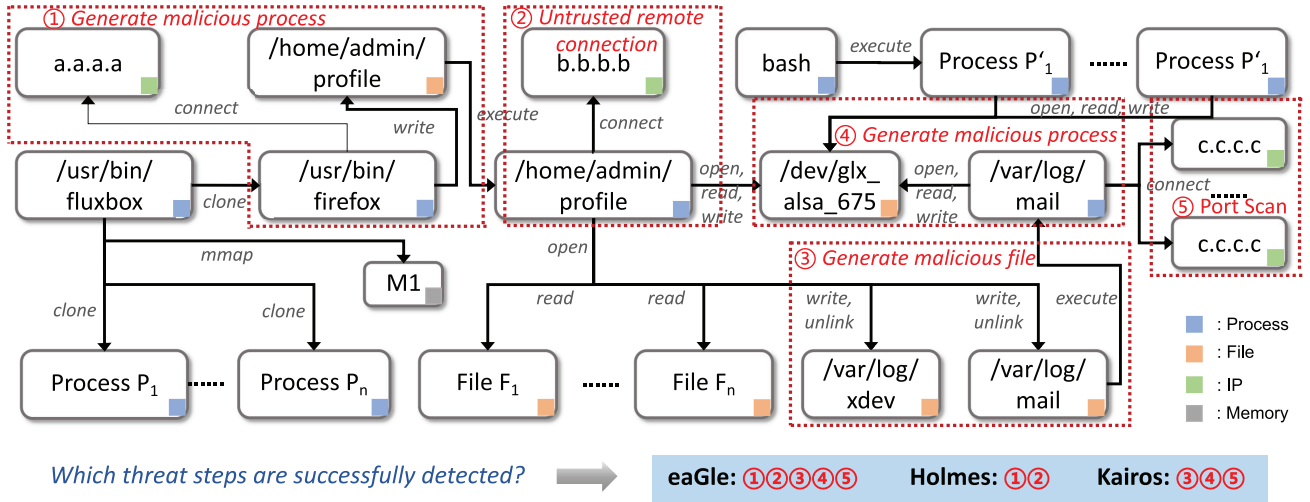


Fig. 1. The toy example of an attack scene.

represents the set of nodes in the graph, signifying entities within the host. For example, a node $v \in V$ could represent a process entity such as “firefox.exe.” E represents the set of directed edges in G . Each edge $e \in E$ indicates the type and direction of information flow between two entities in the host. For instance, an edge from a file node to a process node might signify a “read” operation in the provenance graph G . The functions $\mathcal{M}_v : V \rightarrow \Sigma$ and $\mathcal{X}_e : E \rightarrow \Sigma$ map each node and edge to their respective types from an alphabet Σ . Figure 1 provides an example of a provenance graph generated within a host.

B. Motivation Example

We present the limitations of existing host-based threat scenario reconstruction research and the motivation behind this paper using the example in Figure 1, which is derived from the *Theia* scenario in the DARPA TC project [24]. The figure depicts a data provenance graph consisting of four types of nodes and their connections, which can be divided into benign nodes and anomalous nodes (associated with the following five attack steps):

- **① Generate malicious process.** The attacker exploits a backdoor in Firefox 54.0.1 to plant the file `/home/admin/profile` on the victim host, which then executes with root privileges.
- **② Untrusted remote connection.** The process `/home/admin/profile` establishes a connection to the attacker’s command-and-control (C2) server at `b.b.b.b`.
- **③ Generate malicious file.** A malicious file named `/var/log/mail` is dropped onto the system.
- **④ Generate malicious process.** This file later executes as a process `/var/log/mail` with root-level access.
- **⑤ Port scan.** The attacker performs a port scan against the target `c.c.c.c`.

The goal of host-based threat scenario reconstruction is to identify the complete attack scenario from the raw provenance graph, which should include as many anomalous nodes as possible while minimizing benign nodes, and interpret all attack steps.

Related work can be categorized into two main approaches: anomaly-based detection and rule-based matching. We evaluate a state-of-the-art anomaly-based method Kairos [10] and a rule-based method Holmes [6] using the data in Figure 1. The results show that EAGLE successfully detect all five threat steps, while Holmes and Kairos identify Step ①② and ③④⑤, respectively. Although interpreting the results of anomaly-based host threat detection models remains an unresolved challenge, we conduct an in-depth analysis and attempt to interpret Kairos’ detection results. We find that in Step ① and ②, the process, IP, and file interactions (such as connections and writes) show no significant difference from common benign behaviors on the host, leading Kairos to miss these steps. However, in Step ③ and ④, the privilege escalation operations are uncommon for the host in this example, and in Step ⑤, the behavior of connecting to a large number of ports deviates substantially from normal activity. As a result, Kairos successfully detects Step ③, ④ and ⑤. However, such methods cannot explain attack steps. Meanwhile, the results of rule-based method show that it identifies Step ① and ②. Among them, the Step ① is detected by the rule “Untrusted File Exec” in the *Initial_Compromise* stage, and the Step ② is detected by the rule “CnC” in the *Establish_Foothold* stage. The remaining steps are not detected because the ruleset lacks relevant prior knowledge about them. For instance, in Step ③ and ④, `/var/log/mail` is not included in the blocklist, while in Step ⑤, the ruleset lacks rules pertaining to scanning a large number of ports. Rule-based methods heavily rely on prior knowledge (e.g., blocklists) while in practice, it is challenging to maintain comprehensive prior knowledge, particularly regarding zero-day attacks. Another limitation for rule-based methods is that without sufficient prior knowledge, the reconstructed attack scenarios contain excessive benign nodes (e.g., if a rule designed to detect malicious processes does not specify the process name, it may incorrectly flag benign processes such as `Process P1` in Figure 1 as threats), posing significant challenges for security analysts—even though rule-based methods can explain attack steps.

Based on these findings, we propose a novel algorithmic framework that combines the strengths of both approaches: leveraging rule-based methods for precise anomaly localization and interpretability §IV-A, while using anomaly detection to filter out false positives and false negatives from rule matching §IV-B.

As demonstrated by the APT lifecycle [6], attackers first compromise a host and then establish persistence. Figure 1 confirms this pattern: the attack steps exhibit causal dependencies, ultimately tracing back to the root attack node `/usr/bin/firefox`, forming an attack subtree. To accurately extract this subtree from large-scale provenance graphs, we design an anomaly score propagation-based subtree detection algorithm §IV-C. The core idea is to compute anomaly scores for nodes using both rule matching and anomaly detection, then propagate these scores from leaf nodes toward the root. To prevent numerical explosion (which could mistakenly select benign nodes like `/usr/bin/fluxbox` as the root), we introduce a decay mechanism during propagation. Additionally, we develop a pruning algorithm to eliminate benign nodes (e.g., File $F_1, \dots, \text{File } F_n$), ensuring the subtree contains minimal false positives.

In this case, EAGLE detects Step ① and ② through its fine-grained rule-matching module, identifies other steps via its anomaly detection module, and ultimately pinpoints the specific attack graph while generating attack descriptions for all five steps using its anomalous subtree detection algorithm and coarse-grained rules matching algorithm.

III. THREAT MODEL

EAGLE is a host-based threat detection and scene reconstruction system. We make the following assumptions about the adversary:

- **Targets a single host.** The attacker targets a single host and employs a series of methods to infiltrate. These attack methods could be zero-day, meaning their attack patterns have never been recorded before.
- **Performs Complex and structured intrusion.** The attack methods employed by the attacker consist of multiple steps, which are interrelated and can form a continuous structure.
- **Leaves traces of intrusion.** The attacker's campaigns should, in theory, be detectable, indicating that the behavior of malicious entities should differ from that of benign entities.

In addition to these assumptions about the adversary, we also assume that the provenance generation system is reliable. Like other provenance-based related research, EAGLE does not investigate the correctness of the provenance graph itself. Therefore, we assume the integrity of the data used to construct the provenance graph, specifically that the system audit logs are preserved without tampering throughout attack intervals.

IV. DESIGN

In this section, we present the overall design of EAGLE, which encompasses five main components illustrated in Figure 2 and outlined as follows:

- (§IV-A) ① **Fine-grained Rules Matching:** EAGLE takes the provenance graph, generated by external tools, as input and matches anomalous nodes with fine-grained rules. The fine-grained rules inherently possess explanatory capabilities, so the anomalous nodes they match contribute to both the completeness and interpretability of the ultimately reconstructed attack scenario.
- (§IV-B) ② **Anomaly-based Detection:** EAGLE employs anomaly-based detection models to identify *suspicious nodes* within the same provenance graph used by the fine-grained rules matching component. It operates independently of the Fine-grained Rules Matching module, with both modules separately analyzing the provenance graph to detect anomalous nodes for subsequent processing steps. The contribution of this module lies in detecting attack-related nodes that are not covered by the rules, thereby enhancing the completeness of the reconstructed attack scenario.
- (§IV-C) ③ **Anomalous Subtree Detection:** This component takes the provenance graph as input and calculates the anomaly score of each node in the graph using an iterative approach, resulting in the generation of an anomalous subtree.
- (§IV-D) ④ **Coarse-grained Rules Matching:** In this component, EAGLE uses coarse-grained rules to match the attack types of nodes within the anomalous subtree.
- (§IV-E) ⑤ **Expert Interaction:** For nodes that cannot be matched by coarse-grained rules, EAGLE seeks support from experts to analyze their attack types. The database of coarse-grained rules is updated following expert interaction.

A. Fine-Grained Rules Matching

We employ the term “rules matching” to encompass the category of misuse-based approaches. A rules matching method requires expert knowledge or prior experience with previous attacks to generate attack patterns, which are then transformed into human-readable texts. These texts are referred to as rules, and the collection of rules constitutes the ruleset. The rules utilized by EAGLE are designed based on TTPs [20], which is one of the most popular rulesets concerning host-based intrusion campaigns.

EAGLE is designed for deployment on a host. Initially, EAGLE takes the provenance graph of the host as input, a process facilitated by external tools. For instance, Camflow [21] constructs a unified, whole-system provenance graph $G = (V, E, \mathcal{M}_v, \mathcal{M}_e)$ for a host, ordering events by time and ensuring robust security and comprehensive information flow capture. We initialize three sets: $\mathcal{D}_v : V \rightarrow \Sigma$, $\mathcal{S}_v : V \rightarrow R$, $\mathcal{N}_v : V \rightarrow R$. Here, $\mathcal{D}_v$ records the attack interpretations of each node if it is related to an attack, while $\mathcal{S}_v$ and $\mathcal{N}_v$ store the **Anomaly Score** and **Anomaly Number** of each node, with initial values set to $\{0\}$.

The first component of EAGLE is **Fine-grained Rules Matching**. In this module, EAGLE employs rules to match anomalous edges within the provenance graph G . We extract the fine-grained ruleset RS_f based on TTPs of MITRE's ATT&CK framework [20]. In 2016, the security firm Mandiant

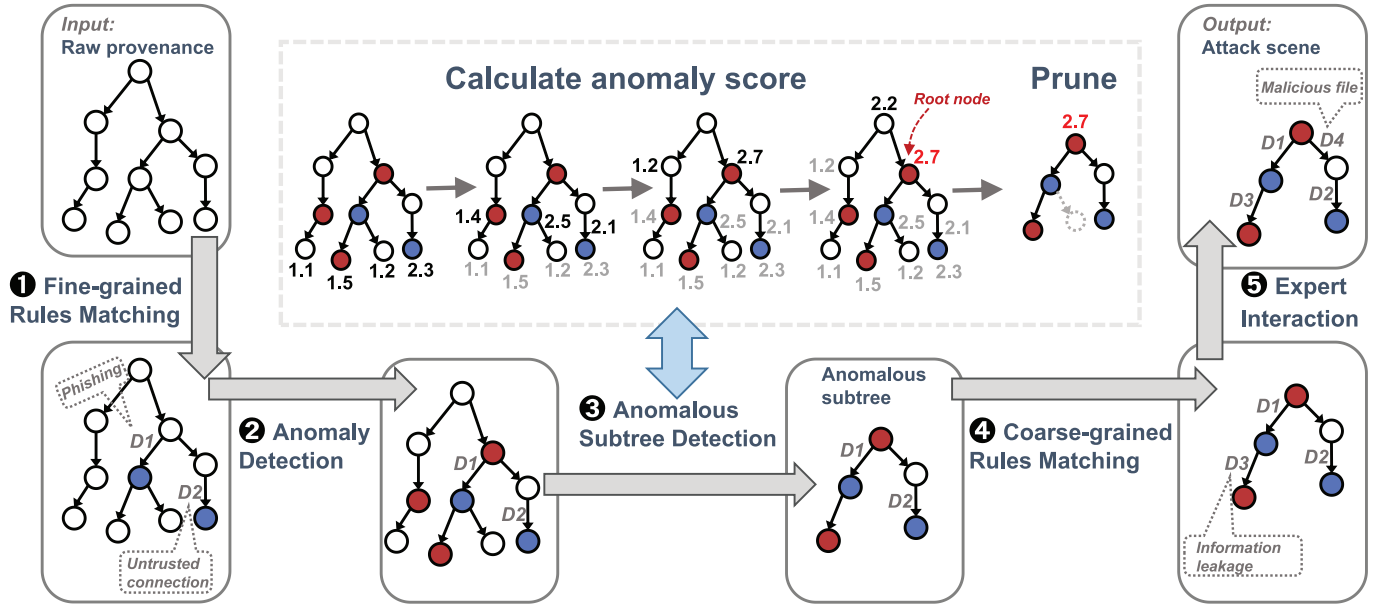


Fig. 2. The overview of components in EAGLE. EAGLE **1** uses fine-grained ruleset to match anomalous nodes in red (§IV-A) and **2** conducts anomaly-based detection to detect anomalous nodes in blue (§IV-B), **3** finds the root node through the anomalous subtree detection algorithm (§IV-C), **4** uses coarse-grained ruleset to match nodes in the subtree other than red nodes (§IV-D), and **5** interprets the remaining unknown nodes through the analysis of experts.

introduced an APT (Advanced Persistent Threat) lifecycle model [25] consisting of seven typical steps: Initial Compromise, Establish Foothold, Escalate Privileges, Internal Recon, Move Laterally, Maintain Presence, and Complete Mission. This APT lifecycle model serves as the basis for various rulesets, including the one used in this study. We provide examples of fine-grained rules in Table III. While there isn't a strict limit to the number of rules, due to space constraints, we present only a subset of the complete ruleset. During the detection phase, the rules matching module scrutinizes each edge $e \in E$ in the provenance graph G . For each edge e , the module compares it with each rule $rs_f \in RS_f$ exhaustively. If e successfully matches a rule rs_f , EAGLE examines the two endpoints of e and determines which endpoint $\hat{v}$ is anomalous according to the rule. The anomaly score of the anomalous node is updated as $\mathcal{S}_v(\hat{v}) = \mathcal{S}_v(\hat{v}) + 1$. Additionally, the attack interpretation of the corresponding node is set based on the matched rule.

Unlike anomaly-based detection, a rules matching approach does not require a training phase for model development. Instead, it directly takes samples as input and iteratively matches each sample with every rule. When a sample is matched with a rule, it is flagged as an anomalous sample. As the anomalous sample is interpreted by the matched rule, it has good readability, facilitating the reconstruction of the complete attack scene. However, the rules matching approach lacks the ability to match anomalous samples whose behavior is not included in the ruleset.

B. Anomaly-Based Detection

Anomaly-based detection comprises various approaches [3], [4], [5], [26] sharing the same detection concept. The workflow of an anomaly-based detection model is depicted in

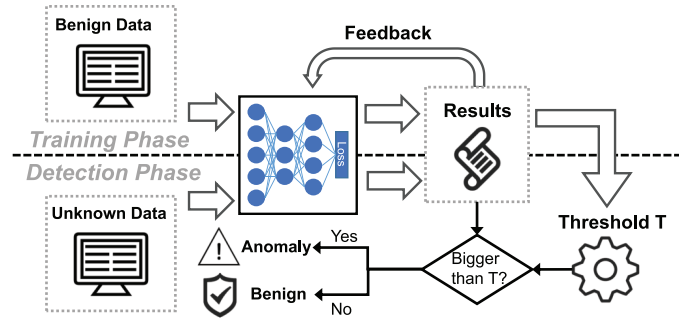


Fig. 3. The general workflow of an anomaly-based detection model.

Figure 3. Typically, an anomaly-based detection approach involves training a model to learn the hidden distribution of benign data. Clustering methods [3] and deep learning models [5], [26] are two commonly employed categories in these approaches. Following the training phase, a threshold is determined through validation using training samples. In the detection phase, samples are passed to the benign model learned during training. The model then generates an anomaly score for each sample. A sample is flagged as anomalous when its anomaly score surpasses the threshold. Anomaly-based detection has the potential to identify unknown attacks by detecting deviations from the benign model [27]. However, it can only raise alerts and cannot specify their specific type.

We note that the concept of anomaly-based detection [27] is easily confused with anomaly detection. Anomaly detection encompasses various methods [28], [29], [30] for detecting anomalies, whereas in this paper the detection methods we focus on are called anomaly-based detection which share the same detection workflow (Figure 3).

The anomaly-based detection component employs an anomaly-based model to calculate the anomaly score of each node $v \in V$. We customize a graph neural network called GAT [19] to learn the hidden distribution of benign nodes and detect anomalous nodes in the provenance graph. As a deep learning model, GAT's primary task is to extract features and assign labels to nodes in the graph. For nodes in the provenance graph, their adjacent edge attributes store substantial causal information, which is why some state-of-the-art host threats detection approaches [5], [31] utilize them to initialize node features. We also adopt a similar method for feature initialization. We quantify the number of different types of nodes and edges as N_n and N_e , respectively. For each node, its feature vector has $N_e + 1$ dimensions. In the first N_e dimensions, each dimension's value indicates the count of a specific type of edge related to the current node. The last dimension denotes the relative time compared to the first node in the graph. This value is calculated by subtracting the timestamps of the current node and the first node. We tailor this feature extraction strategy to capture the temporal and spatial characteristics of each node in the provenance graph. The GAT model takes a $N_e + 1$ dimensional vector as input for each node and outputs a N_n dimensional vector after undergoing a series of linear and nonlinear transformations. Each dimension of the output represents the predictive value for a specific node type, with the total value summing up to 1. We use $Re_v(v_i)$ to denote the abnormal degree of a node v_i . Assuming the value in the dimension corresponding to its actual node type is A_i , $Re_v(v_i)$ is calculated as $1 - A_i$. Being an anomaly-based model, we exclusively employ benign nodes to train the GAT model. The training objective is to minimize the abnormal degrees of these benign nodes. During the detection phase, a node with a high abnormal degree is likely to be a non-benign node that hasn't appeared in the training data.

In summary, the anomaly-based detection model takes the provenance graph G as input and generates the detection results Re_v for each node. For a node $v \in V$, $Re_v(v)$ represents the abnormal degree of v . Consequently, EAGLE updates the anomaly score of each node as $S_v(v) = S_v(v) + Re_v(v)$. The anomaly-based model can be replaced by other appropriate models. We evaluate EAGLE with two other anomaly-based detection models in §V-A to validate its generality. The anomaly-based detection model does not consider semantic information (e.g., process names) but focuses on learning causal relationships in provenance graphs. As EAGLE is a general framework, the anomaly detection module can incorporate models that leverage semantic information (e.g., Kairos [10]). We evaluate this implementation—denoted as EAGLE(Kairos)—in §V-E to assess its scenario reconstruction capability. Furthermore, EAGLE's rules matching module can utilize semantic information from host entities, complementing the detection capacity of the anomaly detection model. In addition to deep learning-based approaches, NLP-based node embedding is another commonly used method for processing data provenance graphs, which aims to represent network nodes as low-dimensional vectors. However, compared to deep learning methods, these approaches lack adaptive learning capabilities and are typically limited to single-task scenarios

[32]. Consequently, most recent studies have shifted toward deep learning models [5], [10], [11], [31], [33] rather than relying on NLP-based node embedding techniques.

C. Anomalous Subtree Detection

Using the anomaly score S_v of each node as input, EAGLE iteratively updates the final anomaly score and the anomaly number $\mathcal{N}_v$ for nodes in the provenance graph. EAGLE selects the most anomalous node as the root node and prunes the corresponding subtree to generate a preliminary attack scenario. This process is referred to as anomalous subtree detection. An illustrative example of the anomalous subtree detection algorithm is provided in Figure 2. It is essential to note that this algorithm works only when there are no cycles in the provenance graph.

We introduce the design rationale of the anomalous subtree detection algorithm here. We devise an iterative aggregation method to amalgamate the anomaly score of each node with its father node. Through this aggregation process, the anomaly score of each node signifies the anomaly score of the entire subtree with it as the root node. To prevent the infinite accumulation of anomaly scores, we incorporate a decay procedure into the algorithm. In addition to the anomaly score, we use the variable **anomaly number** for each node to keep a record of how many anomalous nodes within the subtree with the node as the root. These two values serve as crucial metrics for determining the final selection of the anomalous subtree from two distinct perspectives.

Initially, we calculate the average value of the anomaly score S_v . The computation of the anomalous subtree commences from the leaf nodes of the graph G . We employ a set X to keep track of the current leaf nodes. For every node $v \in X$, X computes its anomaly number $\mathcal{N}_v(v)$. If the anomaly score $\mathcal{N}_v(v)$ surpasses $\alpha * Avr$, node v is identified as a highly suspicious node, and its anomaly number $\mathcal{N}_v(v)$ is incremented by 1. Subsequently, the aggregation procedure begins. The anomaly score and anomaly number of node v are aggregated with its father node $\hat{v}$. Instead of directly adding the values, we introduce a decay factor θ to prevent issues related to infinite accumulation. Through this decay method, when we calculate the anomaly score of a node, the contribution of its child nodes to its anomaly score decreases if its distance from its child nodes is significant long. This approach aligns with real-world scenarios where the relationship between two entities in the host is heavily influenced by their distance. If several other entities are positioned between them, it suggests that they might not belong to the same campaign. After computing and aggregating every node $v \in X$, we remove these nodes from V , record the new leaf nodes into X , and repeat the aforementioned process. Once all nodes in the graph G have been processed, we select the node with the highest sum of S_v and $\mathcal{N}_v$ to be the root node of the anomalous subtree. We define the anomalous subtree as $\hat{G} = (\hat{V}, \hat{E})$ with the root node *Root*.

The subtree needs to be pruned because some nodes in $\hat{G}$ are benign. These benign nodes should be excluded from the final reconstructed scene to avoid false positives. We conduct a breadth-first search starting from the root node *Root*. If the

TABLE I
EXAMPLES OF FINE-GRAINED RULES

Lifecycle step	Rules	Anomalous entity
Initial Compromise	Process A connects to remote IP B and B $\in$ IP black list	B
Establish Foothold	Process A is executed from File B and B $\in$ File black list	B
Escalate Privileges	Process A interacts with SuperUser tools B and A $\in$ Process black list	A
Internal Recon	Process A reads File B and A $\in$ Process black list	A
Move Laterally	Process A connects to remote IP B and A $\in$ Process black list	A

TABLE II
EXAMPLES OF COARSE-GRAINED RULES

Lifecycle step	Rules	Anomalous entity	Description
Initial Compromise	Process A connects to remote IP B	B	Start intrusion
Establish Foothold	Process A is executed from File B	B	Execution of intrusion related shell
Escalate Privileges	Process A interacts with SuperUser tools B	A	The privilege of a malicious process escalates
Internal Recon	Process A reads File B	A	Sensitive information leakage
Move Laterally	Process A connects to remote IP B	A	Lateral movement

TABLE III
OVERVIEW OF THE DARPA TC DATASET

	Dataset	# of benign nodes	# of abnormal nodes	# of edges
Theia	Training set	304,973	0	9,388,492
	Testing set	319,405	25,363	9,426,832
Cadets	Training set	362,645	0	4,347,080
	Testing set	344,316	12,858	4,316,489
Trace	Training set	1,271,939	0	3,572,351
	Testing set	1,143,186	68,265	3,405,673

anomaly number of the current node is 0, we remove this node and its descendant nodes from the anomalous subtree. The rationale behind the pruning procedure is that when a node's anomaly number is 0, the subtree with it as the root node comprises nodes with low anomaly scores. The anomaly score is calculated by both the anomaly-based detection and rules matching modules. Consequently, the nodes within this subtree are likely to be benign. In §V-C, We evaluate whether pruning affects the detection of abnormal nodes in the reconstructed attack scenarios on publicly datasets.

The parameters α and θ can be selected empirically. We explore the impact of these parameters with varying values in §V-E.

D. Coarse-Grained Rules Matching

In this module, each node in the pruned subtree needs to be checked, and interpretations need to be provided for nodes whose $\mathcal{D}_v$ is empty. During the fine-grained rules matching, some nodes have already been interpreted. For nodes with an empty $\mathcal{D}_v$, we employ a rules matching method with a coarse-grained ruleset. Table II presents some examples of coarse-grained rules. Unlike the fine-grained rules in Table III, the coarse-grained rules are more likely to trigger alerts for nodes. To minimize false positives, we do not use the coarse-grained rules as the original rules matching ruleset. Once an anomalous subtree is detected, the detection scope

is substantially reduced. The coarse-grained rules are suitable for providing interpretations of unknown nodes in the subtree because they can match nodes that fine-grained rules cannot.

E. Expert Interaction

The final module requires the interaction of security experts. We assume that in the worst case, some nodes still cannot be matched even with the coarse-grained rules. In such situations, security experts can assist by checking these nodes and providing attack interpretations for them. The interpretations of these nodes can be used to generate new coarse-grained or fine-grained rules, which can then be employed in subsequent detections.

V. EVALUATION

We evaluated EAGLE using three attack scenarios from the DARPA TC dataset [24]. Our evaluation primarily focused on the following six aspects:

- Q1. The detection and reconstruction effects of EAGLE in public datasets and simulated engagements (§V-A, V-C).
- Q2. The generality of EAGLE (§V-A).
- Q3. The performance of each component (§V-C).
- Q4. The runtime performance (§V-D).
- Q5. The influence of parameters (§V-E).
- Q6. The robustness against adaptive attacks (§V-F).

A. Experiments With the DARPA TC Dataset

In this subsection, we evaluate EAGLE with the DARPA TC dataset and compare it with three state-of-the-art threat reconstruction detectors. We replace the anomaly-based detection module with four different models to evaluate the generality of EAGLE as well.

(a) *Dataset*: The DARPA TC dataset is generated during red-team vs. blue-team engagements organized by the DARPA Transparent Computing (TC) program, which aims to develop technologies and prototype systems for real-time detection of advanced threats. There are several threat scenes in the DARPA TC dataset, and we choose three scenes from the third engagement for evaluation. The original dataset consists of system logs. We remove some redundant content and convert the remaining part to the graph format. The provenance data and ground truth are publicly available [24]. We employ the labeling scheme from [5] to annotate anomalous nodes based on the ground truth labels. The overview of the DARPA TC dataset is shown in Table I.

(b) *Anomaly-based detection methods*: We use four approaches for the anomaly-based detection module to evaluate the generality of the proposed framework.

- **GAT model**. This is the model we design for EAGLE. It utilizes the attention mechanism of the GAT model to learn the hidden distribution of different benign nodes.
- **GraphSAGE model**. We adopt the GraphSAGE model from an anomalous nodes detection approach named **threaTrace** [5]. It aggregates information from neighbor nodes to learn different characteristics of benign nodes.

- **Kairos.** Kairos [10] is a SOTA host-based attack reconstruction method. We adopt the anomaly detection module of Kairos, which is a graph autoencoder model.
- **node2vec.** We adopt node2vec [34] to embed the features of each node in the provenance graph. The downstream approach is clustering [35]. We classify nodes into different clusters based on the features extracted by the node2vec approach and detect anomalous nodes using clustering approach.

(c) *Baselines:* We categorize host-based threats reconstruction approaches into two typical categories and select three representative methods to establish the baselines.

- **Rule-based.** The first category comprises rule-based threats detection and reconstruction approaches. We choose Holmes [6] as the representative baseline system. Holmes defines a series of rules for matching anomalous edges in a provenance graph and uses these anomalous edges to construct the attack scene based on the APT kill-chain.
- **Anomaly-based.** The second category consists of anomaly-based methods. We select two state-of-the-art studies, Kairos [10] and NodLink [11], as the baselines. It is important to note that the anomaly-based methods in this category are designed for attack scene reconstruction, which distinguishes them from the attack detection methods in EAGLE's anomaly-based detection module.

(d) *Experimental setup:* We adopt the same dataset partitioning method as [5]. The original dataset for each scenario contains multiple files. We select files containing anomalous behaviors as the test set. Some datasets are removed, including some graphs generated during exceptional incidents such as host outages or shutdowns, while other benign graphs are discarded because we do not need that much training data. Among the remaining files, files without threats are used to train the models, while files containing threats are reserved for evaluation. Both nodes and edges maintain a roughly 1:1 ratio between the training and test sets. Detailed information can be found in Table III. After training the anomaly-based detection model, we use the testing set to evaluate EAGLE. The parameters α and θ are set as 2 and 0.5 based on its performance with varying values.

We repeat the experiments several times and report the average results. The results are analyzed from two perspectives: qualitative analysis and quantitative analysis. Regarding qualitative analysis, we present the reconstructed attack graphs and assess their completeness and interpretability in comparison to the ground truth. For quantitative analysis, we employ *Precision*, *Recall*, *F-Score*, *FPR* (as metric in node-level), and the *graph similarity* [36] (as metric in graph-level), as the evaluation metrics to compare the detection and reconstruction performance with the baselines. Considering the unique characteristics of different detectors, we define specific evaluation metrics for each of them.

- **EAGLE.** The output of EAGLE consists of an anomalous subtree. We categorize the nodes within the subtree as anomalous nodes, while the remaining nodes in the testing set are considered benign. We then compare these

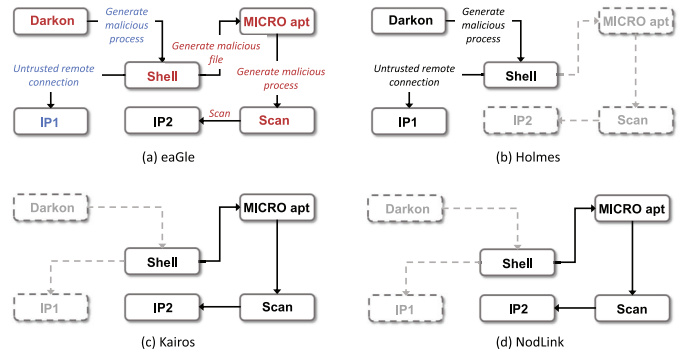


Fig. 4. The reconstructed attack graph of *Theia*.

results with the ground truth nodes to calculate the following evaluation metrics: *True Positives*, *True Negatives*, *False Positives*, and *False Negatives*. Then, we use these metrics to further calculate *Precision*, *Recall*, *FPR*, and *F-Score*. In addition, we utilize *graph similarity* as a metric to assess the effectiveness of scenario reconstruction. We employ the WL subtree graph kernel [36] to calculate the *graph similarity*. Firstly, we calculate the WL subtree graph kernel vectors for the ground truth graph and the abnormal subtree reconstructed by EAGLE. Then, we map the angle between these vectors to a value within the range of 0 to 1 to represent the *graph similarity*. A higher value indicates a better scenario reconstruction effect.

- **Holmes.** Similar to EAGLE, Holmes generates an anomalous graph. Therefore, we apply the same strategy for calculating metrics.
- **Kairos.** The output of kairos consists of several attack summary graphs, which we compare one by one with the ground truth and select the graph with the best restoration effect as the result. Therefore, this somewhat overestimates its detection capability. The subsequent metric calculation process is the same as for other baselines.
- **NodLink.** The output of NodLink is a concise alert provenance graph. The subsequent metric calculation process is the same as for other baselines.

(e) *Results:* Initially, we analyze the quantitative results presented in Table IV. EAGLE exhibits promising performance, outperforming the baselines, particularly when equipped with the GAT model. When compared with Holmes, EAGLE utilizes the anomaly-based detection component to identify alarms missed by rule matching, potentially leading to higher *Recall*. Furthermore, the anomalous subtree detection module hones in on the most likely locations of anomalies, reducing false positives and resulting in a higher *Precision*. The impact of each component on EAGLE's detection performance is assessed in the subsequent subsection. When compared with Kairos and NodLink, the fine-grained rule matching module in EAGLE plays a significant role, enabling it to detect more anomalous nodes, which is reflected in its higher *Recall*.

Additionally, changing the anomaly-based detection model from GAT to GraphSAGE and node2vec does not significantly alter the performance, demonstrating EAGLE's generality.

TABLE IV
THE QUANTITATIVE RESULTS OF EXPERIMENTS IN THE DARPA TC DATASET

Scene	Detector	Precision	Recall	F-Score	FPR	Graph Similarity	TP	TN	FP	FN
Theia	EAGLE(GAT)	0.88	0.92	0.90	0.0096	0.86	23,342	316,351	3,054	2,021
	EAGLE(GraphSAGE)	0.86	0.91	0.88	0.0115	0.85	23,177	315,726	3,679	2,186
	EAGLE(Kairos)	0.87	0.90	0.88	0.0107	0.82	22,933	315,979	3,426	2,430
	EAGLE(node2vec)	0.86	0.85	0.85	0.0117	0.81	21,609	315,681	3,724	3,754
	Holmes	0.85	0.80	0.82	0.0159	0.70	20,378	314,342	5,063	4,985
	Kairos	0.76	0.80	0.78	0.0198	0.74	20,328	313,091	6,314	5,035
	NodLink	0.81	0.75	0.78	0.0139	0.72	19,136	314,974	4,431	6,227
Cadets	EAGLE(GAT)	0.85	0.88	0.86	0.0057	0.88	11,354	342,356	1,960	1,504
	EAGLE(GraphSAGE)	0.84	0.90	0.87	0.0063	0.91	11,630	342,151	2,165	1,228
	EAGLE(Kairos)	0.82	0.88	0.85	0.0071	0.90	11,369	341,871	2,445	1,489
	EAGLE(node2vec)	0.81	0.85	0.83	0.0073	0.91	10,955	341,813	2,503	1,903
	Holmes	0.80	0.82	0.81	0.0076	0.76	10,552	341,713	2,603	2,306
	Kairos	0.71	0.82	0.76	0.0123	0.78	10,558	340,081	4,235	2,300
	NodLink	0.77	0.70	0.73	0.0077	0.75	9,034	341,679	2,637	3,824
Trace	EAGLE(GAT)	0.90	0.95	0.92	0.0061	0.91	65,133	1,136,203	6,983	3,132
	EAGLE(GraphSAGE)	0.91	0.95	0.93	0.0054	0.93	65,020	1,137,013	6,173	3,245
	EAGLE(Kairos)	0.92	0.95	0.93	0.0047	0.92	64,984	1,137,774	5,412	3,281
	EAGLE(node2vec)	0.88	0.89	0.88	0.0072	0.90	60,826	1,134,924	8,262	7,439
	Holmes	0.76	0.85	0.80	0.0157	0.84	58,144	1,125,228	17,958	10,121
	Kairos	0.78	0.76	0.77	0.0127	0.75	52,162	1,128,652	14,534	16,103
	NodLink	0.80	0.72	0.76	0.0106	0.76	49,364	1,131,107	12,079	18,901

Next, we use several case studies to analyse the qualitative results. The simplified reconstructed attack scenes for experiments in EAGLE and baselines are displayed in Figures 4, 5, and 6. For each experimental scene, EAGLE constructs a subtree representing the attack scene, featuring interpretations of nodes, edges, and each attack step.

Figure 4 presents the reconstructed attack graphs of EAGLE and three baselines for the Theia scene. EAGLE reconstructs the complete attack graph as shown in Figure 4(a), with complete interpretation of all 25,363 abnormal nodes. These anomalous nodes are labeled based on the attack description files from DARPA TC [24], and the same applies to the subsequent Cadets and Trace scenarios. The attacker exploits a vulnerability in Firefox 54.0.1. They implant a malicious file, Darkon, into the victim host. Subsequently, a malicious process, Shell, is generated. Shell operates as a RAT (remote access Trojan) and establishes a connection with the attacker's remote server, IP1. The attacker then utilizes Shell to implant another malicious file, MICRO apt. Additionally, a malicious process, Scan, is created to conduct a port scan on IP2. In this scene, the anomaly-based detection component triggers alerts for Shell, MICRO apt, and Scan. The fine-grained rules matching component successfully matches nodes Darkon and IP1, generating their interpretations directly, which are highlighted in blue. The interpretations for the other three steps are generated by the coarse-grained rules matching and are marked in red. Holmes (Figure 4(b)), Kairos (Figure 4(c)), and NodLink (Figure 4(d)) only partially reconstructs the attack graph. Since the ruleset does not include prior knowledge related to Micro apt, the relevant steps do not appear in the reconstructed scenario of Holmes. Holmes can explain the detected attack steps. Neither Kairos nor NodLink detected Darkon and IP1. Our investigation find that this is because the local neighborhood behavior of these two nodes is almost indistinguishable from benign behavior, posing a

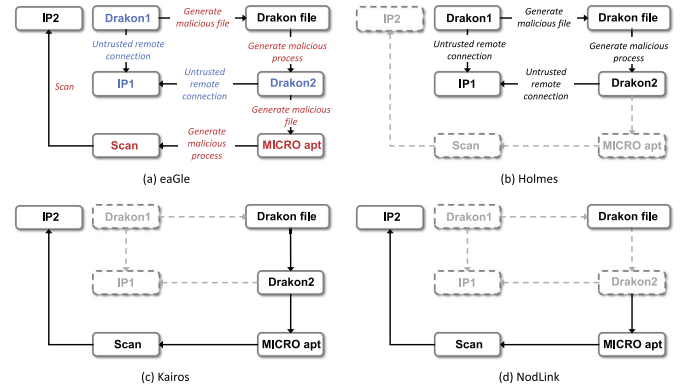


Fig. 5. The reconstructed attack graph of Cadets.

challenge to the anomaly detection model. The above qualitative analysis further demonstrates that EAGLE's combination of two types of methods enables more comprehensive scenario reconstruction and interpretation.

The experimental results for Cadets scenarios are presented in Figure 5. In the Cadets scenario, EAGLE successfully interpret 12,852 out of 12,858 anomalous nodes. The intrusion strategy is somehow similar to the Theia scene. The attacker exploits the vulnerability of Nginx with a malformed HTTP request and generates a Darkon1 implant in the victim host. The drakon implant is connected to attacker's server IP1. The attacker downloads the Darkon file to the victim disk and executes it with root privileges. A RAT Darkon2 starts running and it connects to IP1 again. The attacker downloads the malicious file MICRO apt and generates the malicious process Scan to conduct port scan in IP2. In this scene, the anomaly detection component raises alerts for MICRO apt, and Scan. The fine-grained rules matching component successfully matches nodes Darkon1, Darkon2 and IP1. The interpreta-

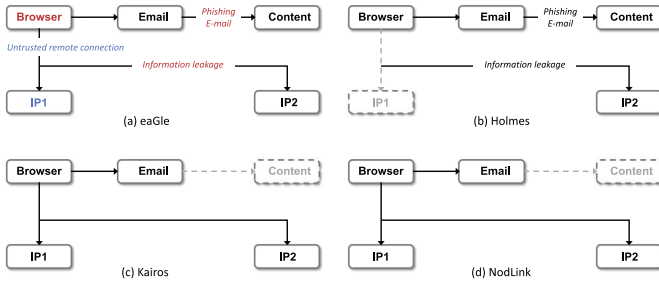


Fig. 6. The reconstructed attack graph of *Trace*.

tions related to these three nodes is directly generated by the fine-grained rules matching. The interpretations of the other steps are generated by coarse-grained rules matching module. As analyzed in the Theia scenario, Holmes (Figure 5(b)) fails to detect some nodes due to limitations in its ruleset. Kairos (Figure 5(c)) and NodLink (Figure 5(d)), on the other hand, misses detection because the behavioral characteristics of certain nodes' neighborhoods are similar to those of benign nodes.

The experimental results for *Trace* scene are presented in Figure 6. In the *Trace* scenario, EAGLE successfully interpret 68,252 out of 68,265 anomalous nodes. The attacker sends a phishing e-mail to the victim. The victim uses *Brower* to open the *Email* process and read the *Content* of the phishing e-mail. The attacker sets a malicious link in the e-mail. The victim clicks the link and opens a website *IP1* which has a form asking for name, e-mail address, and password. The victim unfortunately enters the information and the results are sent to another server *IP2* of the attacker. In this scene, the anomaly detection component raises alerts for *Brower* and the fine-grained rules matching component matches the node *IP1*. The interpretation "Information leakage" is generated by coarse-grained rules matching while the interpretation "Phishing E-mail" is generated by expert interaction. Holmes (Figure 6(b)) fails to detect *IP1* because it is not present in the prior knowledge base. Kairos (Figure 6(c)) and NodLink (Figure 6(d)) misses *content* because the local neighborhood of this node exhibits no difference from normal nodes.

B. Experiments With Simulated Engagements

We deploy an experimental environment in the current network scenario to evaluate the performance of EAGLE in a real-life setting.

(a) *Experimental setup*: Similar to the experimental setup in the DARPA TC dataset, we evaluate the performance of EAGLE in simulated engagements. We assess the performance using four quantitative metrics: *precision*, *recall*, *F-score*, and *graph similarity*, along with qualitative analysis. Due to time and space constraints, we conduct experiments only on EAGLE utilizing the GAT model.

(b) *Experimental schemes*: The experimental environment comprises two hosts, one acting as the attacker and the other as the victim. The attacker executes several intrusion steps to accomplish specific objectives. System logs are collected using an internal tool and stored on the victim's disk. Once

TABLE V
OVERVIEW OF THE SIMULATED ENGAGEMENTS

	Dataset	Duration (h)	# of benign nodes	# of abnormal nodes	# of edges
Scheme 1	Training set	4.1	25,634	0	82,402
	Testing set	2.2	14,203	155	53,031
Scheme 2	Training set	5.3	31,536	0	105,415
	Testing set	2.1	15,964	126	57,942
Scheme 3	Training set	4.8	29,761	0	118,721
	Testing set	2.2	14,612	194	45,418

the intrusion concludes, these system logs are transformed into a provenance graph, forming the testing set. To create the training set, we gather training data while the victim host operates without intrusion campaigns. This benign training set is employed to train the anomaly-based detection module's model. Detailed information can be found in Table V. Due to the limited number of hosts and the duration of attacks, the simulated experiments contain fewer nodes and edges compared to the DARPA TC dataset.

We design three attack schemes, outlined as follows:

- **Scheme 1.** The attacker impersonates a friend of the victim and sends a malicious compressed file, *data.zip*, to the victim through a social software platform called *anonym*. The victim downloads and decompresses the file, obtaining *setup.exe*. Upon running *setup.exe*, the victim triggers a hidden vulnerability. Subsequently, the attacker conducts a port scan, identifies an exploitable port, generates a RAT named *Remote*, and establishes a connection to the attacker's host (IP). The attacker then utilizes *Remote* to download the malicious file *shell.exe* onto the victim's disk. Running *shell.exe* results in another RAT, *Shell*, operating in memory. The attacker employs *Shell* to control the victim host, sending *password.txt* to IP. Finally, the attacker deletes the password and record files.
- **Scheme 2.** The attacker fabricates a link and entices the victim to open it. Upon opening the link, a malicious file, *shell.exe*, is automatically downloaded to the victim's disk. Upon execution, *shell.exe* initiates a RAT named *Shell*, which runs in memory and connects to the attacker's host (IP). The attacker uploads a script, *scan.py*, to the victim. The function of *scan.py* is to search a file folder for a target file with the keyword "password" and store the text in a file called *result.txt*. The attacker then sends *result.txt* to IP and deletes the *scan.py* and *result.txt* files.
- **Scheme 3.** The attacker sends a phishing email to the victim, containing a malicious attachment, *picture*. The victim unwittingly uses the *browser* to read the email and downloads the attachment. Upon opening *picture*, a RAT named *Shell* is generated, connecting to the attacker's host (IP). After gaining control of the victim host, the attacker opens the file *target.txt* and modifies its content. Additionally, the attacker monitors the victim's keyboard input, saving the data in a file named *information*. Finally, the file is sent to IP.

(c) *Results and analysis*: After executing the three experimental schemes, we collect the system logs, convert them into

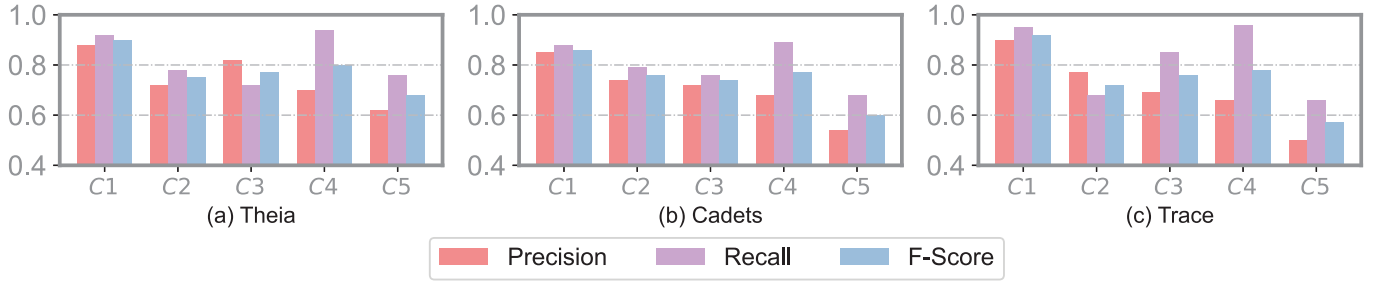


Fig. 7. The results of ablation study.

TABLE VI
THE QUANTITATIVE RESULTS OF EXPERIMENTS WITH SIMULATED ENGAGEMENTS

Scheme	Precision	Recall	F-Score	Graph Similarity
#1	0.84	0.88	0.86	0.87
#2	0.93	0.92	0.92	0.94
#3	0.90	0.96	0.93	0.97

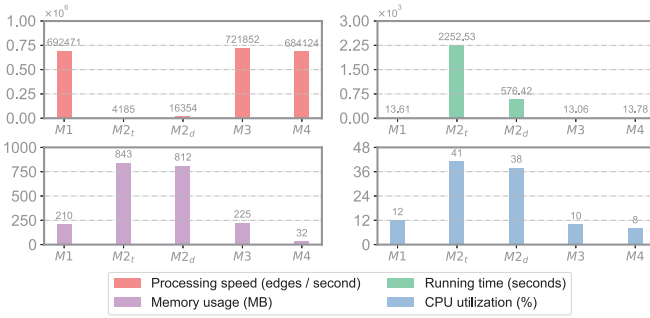


Fig. 8. The results of runtime performance experiments.

the format of a provenance graph, and evaluate EAGLE using them. The quantitative results are shown in Table VI.

For qualitative analysis, EAGLE generates three reconstructed attack graphs. In Scheme 1, Figure 9 presents snapshots after processing in each module.

Scheme 1. The anomaly-based detection module takes the testing data as input and detects several anomalous nodes. As shown in Figure 9 (a), the malicious file `setup.exe`, malicious processes `remote` and `Shell`, and the leaked file `password` are identified as anomalous entities. After the fine-grained rules matching step, malicious files `data.zip` and `shell.exe` are labeled as anomalous. The related edges automatically receive interpretations like “D1: Generate malicious file” and “D2: Generate malicious process” (Figure 9 (b)). Following the anomalous subtree detection module, a subtree with the node `data.zip` as the root node is generated. The edges without interpretations are matched with coarse-grained rules, resulting in five related edges with interpretations such as “D3: Generate malicious process”, “D4: Generate malicious file”, “D5: Untrusted remote connection”, “D6: Information leakage”, and “D7: Untrusted deletion”.

Scheme 2. As shown in Figure 10 (a), after the anomaly-based detection module, the malicious script `scan.py` and

the file `result.txt` are identified as anomalous nodes. The fine-grained rules matching module determines that the file `scan.py` is malicious. The related edge receives the interpretation “D1: Generate malicious process”. Following the anomalous subtree detection module, a subtree with the node `shell.exe` as the root node is generated. Interpretations for the other edges are derived from coarse-grained rules matching: “D2: Generate malicious file”, “D3: Untrusted remote connection”, “D4: Untrusted write”, and “D5: Information leakage”.

Scheme 3. As shown in Figure 10 (b), the anomaly-based detection model detects anomalous nodes, including the processes `browser`, `Shell`, and the file `information`. The fine-grained rules matching module identifies the malicious attachment picture. The related edge is interpreted as “D2: Generate malicious process”. After the anomalous subtree detection module, a subtree with the node `browser` as the root node is generated. Interpretations for the other edges are “D1: Phishing E-mail”, “D3: Untrusted remote connection”, “D4: Untrusted read and write”, “D5: Untrusted write”, and “D6: Information leakage” respectively.

C. Ablation Study

We examine the impact of each component of EAGLE on its performance. Specifically, we evaluate EAGLE using the DARPA TC dataset under four different situations: **C1** with all components, **C2** without the rules matching and anomalous subtree detection components, **C3** without the anomaly-based detection and anomalous subtree detection components, **C4** without the anomalous subtree detection component, and **C5** without anomalous subtree pruning. The experimental settings remains consistent with those of the previous subsection, and the results are presented in Figure 7.

In situations **C2** and **C3**, EAGLE outputs a set of anomalous nodes, and we calculate the evaluation metrics based on the comparison between the reported nodes and the ground truth. There is no significant dominance relationship between these two conditions. In **C2**, the only active module is the anomaly-based detection component, and its performance depends on the training set and the model parameters. In **C3**, the rules matching component replaces the anomaly-based detection module, relying heavily on the ruleset. In **C4**, we combine the reported nodes from the anomaly-based detection and rules matching modules. Consequently, the *Recall* metric increases,

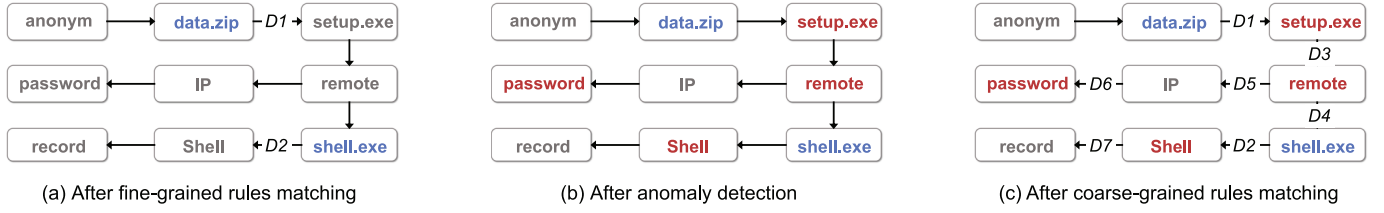


Fig. 9. The snapshots of Scheme 1 after processing in each module.

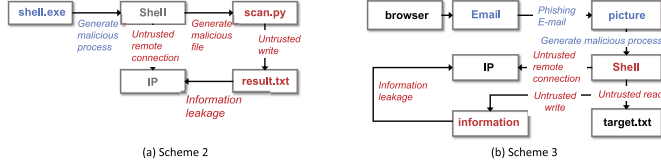


Fig. 10. The reconstructed attack graph of Scheme 2 and Scheme 3.



Fig. 11. Performance with varying parameters.

but the simple aggregation lead to a higher number of false positives, lowering the *Precision* metrics.

C1 represents the condition with all components. Compared with **C3**, **C4** utilizes the anomalous subtree detection component, allowing the concentration of anomalous nodes into a single subtree. This results in the removal of some false positives from the reconstructed attack scene. While some true positives are also eliminated, the results demonstrate that the increase in *Precision* has little impact on *Recall*. The experimental results confirm our intuition behind designing the anomalous subtree detection module. In situation **C5**, we remove the anomalous subtree pruning algorithm and observe that *Recall* does not increase, indicating that the anomalous subtree pruning algorithm does not remove the attack nodes.

D. Runtime Performance

In this subsection, we assess the runtime performance of EAGLE with the aim of identifying the resource consumption of each module. The experiments focus on the *Theia* scene from the DARPA TC dataset, examining three key aspects: processing speed, CPU utilization, and memory usage. The processing speed is quantified as the number of edges processed per second. These experiments are conducted on an Ubuntu 16.04.7 LTS machine equipped with 64 Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz and 128GiB of memory. The implementation runs on a multi-core setup. We evaluate four modules of EAGLE, namely the fine-grained rules matching module (**M1**), anomaly-based detection module (**M2**), anomalous subtree detection module (**M3**), and coarse-grained rules matching module (**M4**). For the anomaly detection module, we test the runtime performance of its training phase and detection phase separately, noted as $M2_t$ and $M2_d$ respectively.

The results, depicted in Figure 8, clearly indicate that the anomaly-based detection module exhibits the highest resource consumption, both in terms of memory usage and CPU utilization. It acts as the runtime bottleneck during execution. The time complexity of the rules matching modules (**M1** and **M4**) is $O(e \cdot r)$, where e represents the number of edges, and r denotes the number of rules. In a real-life scenario, r can

be considered a constant compared to the substantial value of e . Consequently, the time complexity can be simplified further to $O(e)$. The anomalous subtree detection module also operates at $O(e)$ complexity, processing each edge only once. Consequently, the processing speeds of **M1**, **M3**, and **M4** are comparable. In contrast, the anomaly-based detection module employed in this paper's evaluation is a deep learning model, imposing higher demands on computing resources. This leads to lower processing speed, elevated memory usage, and increased CPU utilization. Consequently, it acts as the runtime bottleneck of the entire framework. We further discuss the issue of runtime overhead in §VI. When designing the anomaly detection module algorithm for EAGLE, we adopt the implementation approach from [5], maintaining a fixed-size graph in memory while storing the complete provenance graph on disk. This ensures stable memory usage and CPU utilization even when processing large-scale graphs.

E. Parameters Study

We investigate the influence of the parameters α and θ . We utilize the *Theia* scene from the DARPA TC datasets for evaluation, maintaining one parameter constant at the baseline while testing the other. The baseline values are set at 2 for α and 0.5 for θ .

- **Parameter α .** This parameter is configured to count the number of suspicious nodes in the anomalous subtree detection module (§IV-C). As illustrated in Figure 11(a), EAGLE achieves optimal performance when α is set to 2. A high α value results in some anomalous nodes not being recorded, leading to low Recall. Conversely, a low value generates false positive nodes. Hence, a balanced value is essential to strike a tradeoff.
- **Parameter θ .** This parameter is implemented to prevent infinite accumulation issues when computing the anomaly score of nodes (§IV-C). Figure 11(b) demonstrates an increase in Recall as θ rises from 0.1 to 0.9. This behavior can be attributed to the function of θ : it serves as the decay factor for calculating the anomaly score. A larger

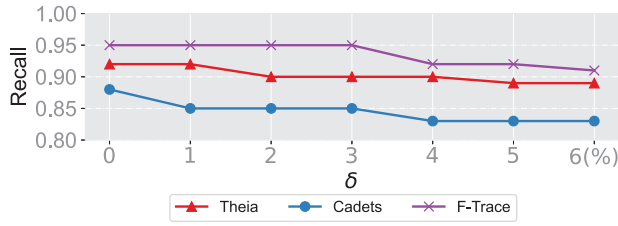


Fig. 12. Results of the adaptive attack experiments.

θ implies a weaker decay effect, leading to more nodes in the anomalous subtree and consequently higher *Recall*. However, an excessively large θ introduces numerous benign nodes into the subtree, decreasing Precision.

The evaluation in this subsection highlights how the parameters of EAGLE influence its detection performance. Further discussions regarding parameter selection are provided in §VI.

F. Robustness

We evaluate the robustness of EAGLE when facing adaptive attacks on the GAT model. Adversaries are categorized based on their level of understanding of the model, ranging from weak [37], [38] to strong [39]. Considering that GAT is a graph neural network model, we refer another GNN-based work [5] to make assumptions about the adversary’s capabilities. We assumed a strong adversary in this scenario, one who possesses detailed knowledge of the GAT model and can modify the data within the provenance graph.

The adversary employs optimization-based evasion attacks to evade EAGLE’s detection. Specifically, the adversary’s objective is to modify the features of a node x related to them in such a way that the modified node x_{new} generates the smallest possible anomaly score detected by EAGLE. Since node features do not affect the anomaly score generated by the rules matching module, the goal is to minimize the value of node x_{new} in the loss function $loss(x_{new})$ of the GAT model. As one of the fundamental principles of optimization-based evasion attacks, the adversary aims to minimize the perturbation between x_{new} and x as much as possible, limited by δ . Therefore, the problem of evading the GAT model can be formalized as the following optimization problem:

$$\operatorname{argmin}_{\tilde{x}}(loss(x_{new})) \quad s.t. \quad \frac{\|x_{new} - x\|_2}{\|x\|_2} < \delta, x_{new} \in \mathbb{N} \quad (1)$$

Results and analysis. We conduct experiments using the DARPA TC dataset. Firstly, we select the attack nodes present in the scenarios reconstructed by EAGLE. We use the optimization equation from Equation 1 to calculate the new features for these attack nodes. Secondly, we input the provenance graph containing these new nodes back into EAGLE to reconstruct the scenarios. We then calculate the values of *Recall*. The experiments are conducted under different settings of δ , and the results are shown in Figure 12. As the constraint gradually relaxes, the impact of the adaptive attack on *Recall* increases. But overall, the impact remains acceptable.

VI. DISCUSSION & LIMITATION

Here, we discuss the limitations of EAGLE, common issues within the research community, and outline our future work.

- (a) **Robustness.** In the real world, attackers might design adversarial attacks [40], [41], [42] to bypass EAGLE. Currently, we haven’t specifically designed defenses against these attacks nor evaluated the system’s robustness. The anomaly-based detection module is particularly vulnerable to adversarial attacks. EAGLE integrates anomaly scores from both the anomaly-based detection and rules matching modules. Hence, attackers would need to compromise both modules. Investigating EAGLE’s robustness against adversarial attacks is a focus of our future research.
- (b) **Polluted training set.** EAGLE employs a GAT anomaly-based detection model to generate node anomaly scores in a provenance graph. This model relies on a benign training set. If the training set contains anomalous nodes, false negative nodes might increase [43], [44]. However, the rules matching component can detect some of these false negatives, offering partial compensation.
- (c) **Evaluation.** We evaluate EAGLE using a public dataset and simulated engagement scenarios. However, both the dataset and deployment schemes are simulations. Obtaining complete real-world attack data is challenging due to intrusions being fragmented, making attack scene reconstruction difficult. Building a comprehensive real-world dataset is essential for advancing the research community, and we aim to achieve this goal in the future.
- (d) **Parameters.** EAGLE relies on two critical parameters: α and δ (§IV-C). We recommend administrators adjust these parameters based on specific situations. Higher α values lead to larger subtrees in anomalous subtree detection, potentially including more intrusion and benign nodes. Lower δ values emphasize child nodes’ impact on the root, aiding long-term threat detection but increasing false positives. Administrators must strike a balance to optimize performance.
- (e) **System overhead and runtime bottleneck.** Real-world deployment concerns center around system overhead. The excessive scale of host data is considered the primary bottleneck in the practical adoption and widespread implementation of current provenance graph-based research [9]. EAGLE, designed to enhance host security, should not hamper other tasks. Presently, there is the potential for optimizing the operational performance of the anomaly-based detection module. We plan to explore alternative, resource-efficient anomaly-based detection methods. Another approach is offloading data processing to the cloud, although challenges exist, such as pruning or compacting data without crucial information loss for attack scene reconstruction [45], which is a focus of our future work.
- (f) **Threat alert fatigue problem.** Addressing the “threat alert fatigue problem” [46] is crucial for the successful implementation of any security research. Both rule-based matching and anomaly detection inevitably

generate false positives. EAGLE mitigates this issue through its Anomalous Subtree Detection module, which locally contextualizes attack scenarios within the provenance graph to filter out potentially false alerts as much as possible. The experimental results of the ablation study can further corroborate this point. As shown in Figure 7, the use of the anomalous subtree detection algorithm (C1) significantly improves precision compared to not using it (C4), indicating a reduction in the number of false positives. When false positives persist in the final reconstructed attack scenario, security analysts can trace whether the node originated from rule matching or anomaly detection modules, then accordingly refine the rules or models to reduce the likelihood of similar false alarms in future operations.

- (g) **Rules.** In practice, without an appropriate rule set, EAGLE will degenerate into anomaly-based detection. We recommend that users refer to public knowledge bases such as TTPs [20] for the design of rule sets.

VII. RELATED WORK

EAGLE is specifically engineered for reconstructing attack scenes within a host, leveraging both rules matching approaches and anomaly-based detection models. We explore the related work in these domains.

A. Anomaly-Based Detection Approaches

Before reconstructing an attack scene, it is essential to detect the attack steps. These detection methods typically fall into two categories: rule-based (or misuse-based) and anomaly-based. anomaly-based detection approaches rely on identifying anomalies within attack steps. Unicorn [3] introduces an anomalous provenance graph detection method using graph kernel to extract evolving features from the provenance graph. ProvDetector [4] identifies anomalous paths by labeling nodes with anomaly scores, calculated based on node frequency, extending this to path anomaly score calculations. Kairos [10] designs a graph autoencoder model to learn a provenance graph's temporal evolution, compute the anomalous degree, and reconstruct the attack footprints. ThreaTrace [5] employs the GraphSAGE model [47] to learn benign node behavior and detect anomalous nodes at the node level. NodLink [11] is also an anomaly-based PIDS which detects intrusion in node level. It is the first PIDS to conduct an open-world evaluation. SHADEWATCHER [48] transforms system entity interactions into a recommendation system format, predicting system entity preferences on interactive entities to classify anomalies. IPG [49] utilizes an autoencoder model to detect anomalous graphs. CONAN [50] introduces a three-phase detection model, concentrating attack campaigns into phases and distinguishing malicious from benign behaviors. CONAN maintains the current states of processes and files for detection, similar to finite state automata (FSA). ATLAS [51] employs a supervised model, extracting attack and non-attack sequences from provenance graphs to learn sequence characteristics, utilizing the LSTM [52] model for precise understanding of attack sequences. DistDet [53] trains a

Hierarchical System Event Tree (HST) model to categorize auditing events based on properties and filters false alarms according to alarm properties. Watson [54] proposes inferring and aggregating the semantics of audit events to abstract behavior, which can bridge the gap between low-level audit events and high-level system behaviors. Flash [33] integrates embedding with GNN to learn benign node behaviors and effectively encode both local and global graph structures into expressive node embeddings. Trec [31] explores identifying APT tactics/techniques—a task typically reliant on rule-based methods with high false positives. To overcome limited threat samples and labeling challenges, Trec first detects anomalous nodes (NOIs), samples tactic-relevant subgraphs, and applies few-shot Siamese learning for APT identification with minimal samples.

These anomaly-based detection methods have a common limitation: they cannot interpret detected anomalies. Moreover, the challenge of modeling all benign behaviors often leads to false positives, making it difficult to reconstruct the attack scene using just one anomaly-based approach.

B. Rules Matching Approaches

Rule-based approaches necessitate a database of rules, which can be generated from expert experience, technical reports, and other sources. Holmes [6] conducts the initial study utilizing rules to match anomalous edges in a provenance graph, basing these rules on TTPs. Another notable rules matching approach is Poirot [7], which operates at the graph level, enabling more accurate detection of anomalous entities compared to Holmes. HINIT [55] proposes combining diverse network threat intelligence to create a heterogeneous graph, while EXTRACTOR [56] employs natural language processing techniques to extract attackers and victims from network threat intelligence, constructing rules for reconstructing attack scenarios. Pagoda [57] employs a rule database to detect host-based intrusions, considering both detection accuracy and time. Reference [16] proposes studying subgraphs in the provenance graph, embedding these subgraphs into vectors, and then employing GNN models for supervised training and detection. ALCHEMIST [58] proposes combining system logs and software logs, utilizing the fine-grained nature of software logs compared to system logs. SLEUTH [59] uses credibility labels and confidentiality labels to measure node credibility and data importance, updating these labels while backtracking and determining anomaly occurrences based on their values. MORSE [60], a subsequent work under the SLEUTH framework, addresses the problem of dependency explosion when backtracking the behavior of attackers.

However, these rule-based approaches heavily depend on the correctness and completeness of the rules, limiting their ability to reconstruct zero-day attack steps.

C. Hybrid Approaches

Hybrid methods are quite common in the field of network intrusion detection. Reference [17] proposes an algorithm that combines the C4.5 decision tree model and one-class SVM model to detect intrusions in network traffic data. Reference

[18] proposes a connection detection system, which uses association rule mining and a misuse module to classify benign and attack connections. However, hybrid methods are relatively less common in detecting host-based attacks and reconstructing the attack scenario. APT-KGL [61] generates virtual APT training samples from open threat knowledge, models contextual information of system entities in a HPG (Heterogeneous Provenance Graph), and learns entity representations within the HPG. APT-KGL takes advantage of both knowledge database and deep learning model. However, its design objective is to detect abnormal entities rather than reconstruct the attack scenario.

VIII. CONCLUSION

We present EAGLE, a general framework designed for reconstructing the attack scene on a host. The core components of EAGLE consist of anomaly-based detection and rule matching, which are versatile and can be substituted with various established methods. To minimize false positives, we introduce an innovative algorithm for detecting anomalous subtrees. Additionally, we propose two types of rules, ensuring detailed interpretations of the final reconstructed scene. We assess EAGLE using three public datasets and three simulated engagements. The results demonstrate its effective performance in reconstructing attacks, both qualitatively and quantitatively.

REFERENCES

- [1] S. Singh, P. K. Sharma, S. Y. Moon, D. Moon, and J. H. Park, "A comprehensive study on APT attacks and countermeasures for future networks and communications: Challenges and solutions," *J. Supercomput.*, vol. 75, no. 8, pp. 4543–4574, Aug. 2019.
- [2] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1851–1877, 2nd Quart., 2019.
- [3] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.
- [4] Q. Wang et al., "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2020.
- [5] S. Wang et al., "THREATTRACE: Detecting and tracing host-based threats in node level through provenance graph learning," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 3972–3987, 2022.
- [6] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, May 2019, pp. 1137–1152.
- [7] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "POIROT: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1795–1812.
- [8] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proc. 19th ACM Symp. Oper. Syst. Princ.*, Oct. 2003, pp. 223–236.
- [9] M. A. Inam et al., "SoK: History is a vast early warning system: Auditing the provenance of system intrusions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2023, pp. 2620–2638.
- [10] Z. Cheng et al., "Kairos: Practical intrusion detection and investigation using whole-system provenance," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 3533–3551.
- [11] S. Li et al., "NODLINK: An online system for fine-grained APT attack detection and investigation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024, pp. 1–18.
- [12] A. Nadeem et al., "SoK: Explainable machine learning for computer security applications," in *Proc. IEEE 8th Eur. Symp. Secur. Privacy (EuroS&P)*, Jul. 2023, pp. 221–240.
- [13] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Erreira, A. Gupta, and L. Z. Granville, "AI/ML for network security: The emperor has no clothes," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA, 2022, pp. 1537–1551.
- [14] D. Han et al., "DeepAID: Interpreting and improving deep learning-based anomaly detection in security applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2021, pp. 3197–3217.
- [15] F. Dong et al., "Are we there yet? An industrial viewpoint on provenance-based endpoint detection and response tools," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2023, pp. 2396–2410.
- [16] E. Altinisik, F. Deniz, and H. Taha Sencar, "ProvG-searcher: A graph representation learning approach for efficient provenance graph search," 2023, *arXiv:2309.03647*.
- [17] G. Kim, S. Lee, and S. Kim, "A novel hybrid intrusion detection method integrating anomaly detection with misuse detection," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1690–1700, Mar. 2014.
- [18] D. Barbara, "Adam: Detecting intrusions by data mining," in *Proc. IEEE Workshop Inf. Assurance Secur.*, vol. 1, Jun. 2001, p. 1100.
- [19] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *Stat.*, vol. 1050, no. 20, pp. 1–12, 2017.
- [20] T. M. Corp. (2023). *Att&ck*. [Online]. Available: <https://attack.mitre.org/>
- [21] T. Pasquier et al., "Practical whole-system provenance capture," in *Proc. Symp. Cloud Comput.*, Santa Clara, CA, USA, Sep. 2017, pp. 405–418.
- [22] A. Bates, D. Tian, R. B. K. Butler, and T. Moyer, "Trustworthy whole-system provenance for the Linux kernel," in *Proc. USENIX Conf. Secur. Symp. (SEC)*, Austin, TX, USA, Aug. 2015, pp. 319–334.
- [23] D. J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler, "Hi-Fi: Collecting high-fidelity whole-system provenance," in *Proc. 28th Annu. Comput. Secur. Appl. Conf.*, Dec. 2012, pp. 259–268.
- [24] A. D. Keromytis. (2018). *Transparent Computing Engagement 3 Data Release*. [Online]. Available: <https://github.com/darpa-i2o/Transparent-Computing/blob/master/README-E3.md>
- [25] (2016). *Mandiant: Exposing One of China's Cyber Espionage Units*. [Online]. Available: <https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf>
- [26] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, vol. 5, 2018, p. 2.
- [27] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Secur. Privacy*, May 2010, pp. 305–316.
- [28] G. Apruzzese et al., "The role of machine learning in cybersecurity," *Digit. Threats, Res. Pract.*, vol. 4, no. 1, pp. 1–38, Jul. 2022.
- [29] S. Jose, D. Malathi, B. Reddy, and D. Jayaseeli, "A survey on anomaly based host intrusion detection system," *J. Phys., Conf. Ser.*, vol. 1000, Apr. 2018, Art. no. 012049.
- [30] J. Jabez and B. Muthukumar, "Intrusion detection system (IDS): Anomaly detection using outlier detection approach," *Proc. Comput. Sci.*, vol. 48, pp. 338–346, Jan. 2015.
- [31] M. Lv et al., "TREC: APT tactic / technique recognition via few-shot provenance subgraph learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Dec. 2024, pp. 139–152.
- [32] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
- [33] M. Ur Rehman, H. Ahmadi, and W. Ul Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2024, pp. 3552–3570.
- [34] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 855–864.
- [35] L. Kaufman and P. J. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis*. Hoboken, NJ, USA: Wiley, 2009.
- [36] N. Shervashidze, P. Schweitzer, E. J. V. Leeuwen, K. Mehlhorn, and K. Borgwardt, "Weisfeiler-Lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, no. 77, pp. 2539–2561, Feb. 2011.
- [37] G. Apruzzese, H. S. Anderson, S. Dambra, D. Freeman, F. Pierazzi, and K. Roundy, "Real attackers don't compute gradients: Bridging the gap between adversarial ML research and practice," in *Proc. 1st IEEE Conf. Secure Trustworthy Mach. Learn. (SaTML)*, Feb. 2023, pp. 339–364.
- [38] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "FlowLens: Enabling efficient flow classification for ML-based network security applications," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.
- [39] D. Arp et al., "Dos and don'ts of machine learning in computer security," in *Proc. USENIX Secur. Symp.*, 2022, pp. 3971–3988.

- [40] D. Zügner, A. Akbarnejad, and S. Günnemann, "Adversarial attacks on neural networks for graph data," in *Proc. 24th ACM SIGKDD Int. Conf.*, Aug. 2018, pp. 6246–6250.
- [41] B. Wang and N. Z. Gong, "Attacking graph-based classification via manipulating the graph structure," in *Proc. 26th ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, London, U.K., 2019, pp. 2023–2040.
- [42] L. Sun et al., "Adversarial attack and defense on graph data: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 8, pp. 7693–7711, Aug. 2022.
- [43] G. Apruzzese, P. Laskov, and A. Tastemirova, "SoK: The impact of unlabelled data in cyberthreat detection," in *Proc. IEEE 7th Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2022, pp. 20–42.
- [44] T. V. Ede et al., "DEEPCASE: Semi-supervised contextual analysis of security events," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 522–539.
- [45] T. Zhu et al., "General, efficient, and real-time data compaction strategy for APT forensic analysis," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3312–3325, 2021.
- [46] (2020). *How Many Alerts is Too Many To Handle?*. [Online]. Available: <https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html>
- [47] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. NIPS*, 2017, pp. 1024–1034.
- [48] J. Zengy et al., "SHADEWATCHER: Recommendation-guided cyber threat analysis using system audit records," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2022, pp. 489–506.
- [49] Z. Li, X. Cheng, L. Sun, J. Zhang, and B. Chen, "A hierarchical approach for advanced persistent threat detection with attention-based graph neural networks," *Secur. Commun. Netw.*, vol. 2021, pp. 1–14, May 2021.
- [50] C. Xiong et al., "Conan: A practical real-time APT detection system with high accuracy and efficiency," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 551–565, Jan. 2022.
- [51] A. Alsaheel et al., "ATLAS: A sequence-based learning approach for attack investigation," in *Proc. 30th USENIX Secur. Symp. (USENIX Secur.)*, Jan. 2021, pp. 3005–3022.
- [52] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
- [53] F. Dong et al., "Distdet: A cost-effective distributed cyber threat detection system," in *Proc. 32th USENIX Secur. Symp. (USENIX Secur.)*, 2023, pp. 6575–6592.
- [54] 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 *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.
- [55] J. Zhao, Q. Yan, X. Liu, B. Li, and G. Zuo, "Cyber threat intelligence modeling based on heterogeneous graph convolutional network," in *Proc. 23rd Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, 2020, pp. 241–256.
- [56] K. Satvat, R. Gjomemo, and V. Venkatakrishnan, "EXTRACTOR: Extracting attack behavior from threat reports," in *Proc. IEEE Eur. Symp. Security Privacy (EuroS&P)*, Sep. 2021, pp. 598–615.
- [57] 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 Trans. Dependable Secure Comput.*, vol. 17, no. 6, pp. 1283–1296, Nov. 2020.
- [58] L. Yu et al., "ALchemist: Fusing application and audit logs for precise attack provenance without instrumentation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.
- [59] M. N. Hossain et al., "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *Proc. USENIX Conf. Secur. Symp.*, Vancouver, BC, Canada, Aug. 2017, pp. 487–504.
- [60] M. N. Hossain, S. Sheikhi, and R. Sekar, "Combating dependence explosion in forensic analysis using alternative tag propagation semantics," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 1139–1155.
- [61] T. Chen et al., "APT-KGL: An intelligent APT detection system based on threat knowledge and heterogeneous provenance graph learning," *IEEE Trans. Dependable Secure Comput.*, early access, Dec. 26, 2022, doi: [10.1109/TDSC.2022.3229472](https://doi.org/10.1109/TDSC.2022.3229472).