A Comparative Analysis of Difficulty Between Log and Graph-Based Detection Rule Creation

Matt Jansen
Oregon State University
matt.jansen7@pm.me

Rakesh Bobba
Oregon State University
rakesh.bobba@oregonstate.edu

Dave Nevin
Oregon State University
dave.nevin@oregonstate.edu

***Abstract*—Provenance-based Intrusion Detection Systems (PIDS) are threat detection methods which utilize system provenance graphs as a medium for performing detection, as opposed to conventional log analysis and correlation techniques. Prior works have explored the creation of system provenance graphs from audit data, graph summarization and indexing techniques, as well as methods for utilizing graphs to perform attack detection and investigation. However, insufficient focus has been placed on the practical usage of PIDS for detection, from the perspective of end-user security analysts and detection engineers within a Security Operations Center (SOC). Specifically, for rule-based PIDS which depend on an underlying signature database of system provenance graphs representing attack behavior, prior work has not explored the creation process of these graph-based signatures or rules. In this work, we perform a user study to compare the difficulty associated with creating graph-based detection, as opposed to conventional log-based detection rules. Participants in the user study create both log and graph-based detection rules for attack scenarios of varying difficulty, and provide feedback of their usage experience after the scenarios have concluded. Through qualitative analysis we identify and explain various trends in both rule length and rule creation time. We additionally run the produced detection rules against the attacks described in the scenarios using open source tooling to compare the accuracy of the rules produced by the study participants. We observed that while the graph-based creation exercises required more effort to complete, they resulted in higher interpretability and lower false positives as compared to log-based methods.**

I. Introduction

With the increasing prevalence of technology and networked devices in our daily lives, there is also an increased rate of cybercrime and computer hacking operations. In just this past year, global cyberattacks have increased by 38%, and there has been a 112% increase in demand for access brokerage services on darknet marketplaces [1, 2]. Included in this trend is activity from Advanced Persistent Threat (APT) groups, well-funded and sophisticated hacking groups engaged in prolonged network intrusion, data exfiltration, and cyber espionage [3]. Detecting intrusions performed by APT groups can be difficult since, oftentimes, their goal is to lay low and evade detection. However, it is also clear that organizations have learned to protect themselves by utilizing real-time cyber threat detection. This is evident from the constant increase in intrusions detected internally and the global median dwell time decreasing over the past year [4].

Detecting APT group activity can be difficult for a variety of reasons. For one, they are well-funded adversaries who may be able to execute previously unseen techniques to evade detection. For example, in just this past year, Mandiant reported that over 50% of the initial infection vectors they were able to identify belonged to either usage of a software exploit (37%) or through supply chain compromise (17%) [4]. Given how much money cybercrime costs organizations annually, accurately detecting APT activity has become an important focus in industry and academia. Although this is the case, research has shown how modern Endpoint Detection and Response (EDR) software still fails to detect a majority of emulated APT attacks. For instance, Karantzas et al. utilized DLL-sideloading, a technique known since 2017, to bypass 9 out of 11 modern EDR solutions and run a malicious payload [5, 6]. It is clear that conventional EDR solutions are not sufficient for detecting sophisticated adversaries and, in some cases, not even for detecting well-known and understood threats.

To further increase visibility into networked systems to facilitate improved detection and forensic analysis, recent works have proposed tracking the execution flow of a computer system as a provenance graph [7, 8]. Constructing a view of the system's execution in the form of a directed acyclic graph allows for the tracking of causal dependencies between system entities and enables enhanced correlation of system behaviors. Additionally, when investigating cyber attacks, converting system logs into a provenance graph may aid in the analyst's interpretation of the attack, resulting in better incident response capabilities. With the benefits that system provenance graphs can provide to threat detection and investigation, integrating system provenance graphs into cybersecurity has demonstrated to be a very promising research direction. So much so that dozens of proposals have been made in academia for cyber threat detection software that utilizes system provenance graphs over the past 10 years [9, 10].

However, if Provenance-based Intrusion Detection Systems (PIDS) have been explored in academia for several years now with primarily successful results, then why aren't they being seen breaking into industry more? This issue has been emphasized given the analysis performed by Feng et al., where they had found that most industry representatives agree that PIDS has the potential to outperform EDR; however the additional computational and analyst workload overhead is

Workshop on SOC Operations and Construction (WOSOC) 2024
1 March 2024, San Diego, CA, USA
ISBN 979-8-9894372-3-8
https://dx.doi.org/10.14722/wosoc.2024.23004
www.ndss-symposium.org

too great [11]. Understanding the overheads associated with PIDS deployments, in addition to the extent of benefits it can provide, can help facilitate the integration of PIDS into enterprise networks. That being said, most recent literature surrounding the reduction of overheads associated with PIDS focus on the detection and investigation algorithms, rather than PIDS usage from the perspective of an organization's SOC analyst or detection engineer.

In this paper, we focus on the gap in academic literature surrounding the comparison of graph-based intrusion detection systems (specifically PIDS) to conventional log-based detection techniques. Specifically, this work compares the difficulty associated with creating accurate detection rules for graph-based detection methods with those of log-based detection methods. This was accomplished through a user study where participants built intrusion detection rules for a variety of attack scenarios, and provided feedback based on various aspects of the rule-creation exercises. From the results of the user study, we define metrics for rule creation difficulty and accuracy, and aggregate scores to synthesize observations supported through feedback provided by the participants. To the best of our knowledge, this is the first work addressing PIDS usage from the perspective of its end users through a user study, where the study participants actively builds rules for a rule-based PIDS. This work is also the first to address the comparison of graph-based and log-based endpoint detection through a user study, where the study participants build detection rules for both types of rule-based detection.

Through this research, this paper makes the following contributions:

- We define and justify a set of metrics for the purpose of assessing the difficulty of creating intrusion detection rules for a rule-based IDS.
- We perform a qualitative analysis using the previously discussed metrics to assess the comparative difficulty of creating accurate graph and conventional log-based IDS rules for endpoints.
- We outline numerous areas for potential future research efforts related to the integration of PIDS into enterprise networks.

The remainder of this paper is organized as follows. Section II discusses critical background details related to system provenance graphs and PIDS. Section III details works related to this research effort. Section IV outlines the user study performed as part of this research, and section V goes over the subsequent analysis of the user study results. Finally, section VI discusses takeaways from our research and future work.

II. Background

This Section will review the information necessary to follow the remaining sections of this document, including system provenance graphs, PIDS components, and PIDS detection methods.

A. System Provenance Graphs

PIDS software depends on system audit data first being converted into a graph format, which are referred to as system provenance graphs. These graphs depict the data provenance of endpoint audit records capturing the the origin of a data point, and all interactions made with that data point to determine how it arrived at its current state [12]. Applying data provenance to system audit records in a graph format results in an understanding of the full history of execution of a given endpoint. Specifically, graph nodes represent system entities (e.g., processes, files, network sockets, and memory segments), while the edges represent events connecting the two entities (e.g., system calls). Converting audit records into system provenance graphs strengthens downstream analysis tasks such as correlation between system events and casual analysis. An example of a system provenance graph is shown in Figure 1.

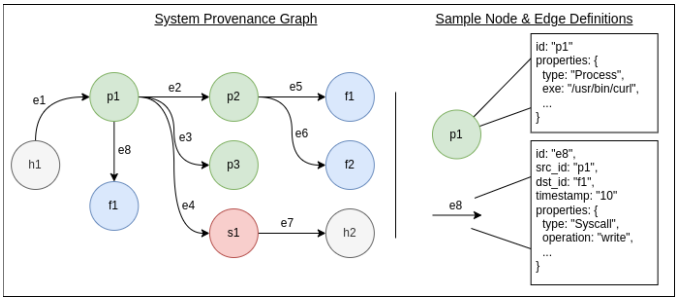

Fig. 1: Example System Provenance Graph

Due to the advanced capabilities associated with system provenance graphs, dozens of threat detection and attack investigation approaches have been proposed in the literature that utilize system provenance graphs as a medium for detection [9, 10]. These systems which perform provenance-based attack detection are called *Provenance-based Intrusion Detection Systems*, or *PIDS* for short.

B. PIDS Components

PIDS are typically composed of 3 components - provenance graph collection, data management, and threat detection [9]. The first module, the provenance graph collection module, is responsible for acquiring endpoint logs and translating them into a system provenance graph. Once a provenance graph is generated, the data management module is responsible for storing and interfacing with the graph — this includes choosing how to store the graph to disk, how to query the graph, and how to prune the graph over time. Once the graph has been processed into its final form, the threat detection module performs analyses on the system provenance graph to detect malicious behavior.

The three main classes of threat detection used by PIDS include signature-based, anomaly-based, and tag-based detection [13, 14]. Signature-based detection entails discovering malicious system behavior by matching signatures of malicious behaviour to the system provenance graph. Typically this is implemented as sub-graph pattern matching, where the signature database is a set of provenance sub-graphs representing malicious behaviors. Additionally, anomaly-based detection utilizes the system provenance graph's structure and the attributes of each entity to create a baseline behavioral model. An anomaly-based PIDS would then monitor for moments when the system

provenance graph deviates far enough from the baseline model to be flagged as anomalous, and raises an alert. Finally, tag-based detection assign tags to nodes and edges according to a policy, and perform tag propagation algorithms to identify malicious behaviors and trace attack paths. Signature and anomaly-based detection are similar to what is seen in conventional endpoint detection software except for they act on graphs, rather than log files.

III. Related Works

In this section we discuss past work relevant to the proposed work organized into two themes. We first discuss the issue of alert fatigue plaguing internal security operations teams, and prior works which utilize PIDS to enable improved alert triage and overall workload reduction. We also review prior work where user studies are performed to gain a better understanding of security operations from various points of view.

A. PIDS Analyst Workload Reduction

In Forrester's 2020 State of Security Operations report [15], it was found that internal security operations teams receive an average of over 11,000 alerts per day. From this extreme workload placed on the shoulders of security analysts, alert fatigue can plague internal security teams by driving attention and effort away from true positive alerts. Alert fatigue is a phenomenon in which an environment is created where, due to the large amount of false positive alerts generated by conventional tooling, true and false positive alerts become difficult to distinguish [16]. In an effort to mitigate alert fatigue, several works have been proposed which aid in the investigation and triage of alerts generated by both conventional security tooling and PIDS. The need for these tools has been demonstrated through user studies performed by Dong et al. [11], where it was found that alarm triage and alarm interpretation cost were one of several deciding factors for implementing PIDS into organizations' networks.

Attack investigation often follows the action of reconstructing attack behaviors found within the system provenance graph after a detection has taken place, or anomalous behavior has been identified. For instance, in Zeng et al. [17], the authors aggregate the semantics of system events to define high-level behavioral patterns which can be clustered to identify anomalous behaviors. Additionally, Hossain et al. [18] presented SLEUTH – a real-time method for detecting and reconstructing attack scenarios through the application and propagation of tags onto system entities. By framing the analysis of multi-stage attacks as a community discovery problem, Pei et al. [19] were also able to perform attack reconstruction by extracting behaviors into communities, identifying communities associated with malicious behaviors, and visualizing the resulting multi-step attack chain.

Alarm triage is another critical downstream analysis task which can aid in mitigating the alarm fatigue problem by deciding which alarms are most likely to be true positives. An example of alarm triage applied to system provenance graphs includes work by Hassan et al. [20], where the authors utilize anomaly detection to assign anomaly scores to system entities and diffuse the scores throughout the graph, ultimately resulting in scores which can be used for triaging. Additional work by Hassan et al. [21] utilizes alerts generated by conventional security tooling to build an alert dependency graph, which is then summarized into several sequences of techniques which map back to MITRE's ATT&CK framework[1], where each sequence is given an alert triage score. Furthermore, prior work by Liu et al. [22] analyzes the rareness of system event occurrences through backwards and forwards tracking within the system provenance graph to prioritize events for further analysis.

The work presented in this paper focuses specifically on rule-based PIDS, and the difficulty of interfacing with such technologies in comparison to conventional security tooling. Specifically, we focus on the difficulty associated with creating detection rules so that detection may take place in the future, in a comparative manner between conventional log-based and graph-based detection software.

B. User Studies of Cybersecurity Operations

In recent literature, there's been an increased focus on improving the effectiveness and efficiency of security operation centers within organizations. This emphasis is the result of the realization of the critical importance of an organization's Security Operations Center (SOC) in the prevention and detection of cyber attacks. To accomplish this, these works run user studies with members of real organizations' security teams as participants. These studies tend to take one of two approaches: (1) passive studies, which ask questions without requiring a technical task to be completed, often in the form of interviews or questionnaires; and (2) active studies, which have the participants completing a security-related technical task, such as rule creation or alarm investigation.

With regard to passive SOC studies, in Kokulu et al. [23], the authors conducted several in-depth interviews to try to further understand the identify issues that occur within SOCs. From these 1-on-1 interviews, several factors - including low network/infrastructure visibility, ineffective phishing training, lackluster SOC performance metrics, and lack of impact from false positives - were identified as the most frequently seen. Alahmadi et al. [24] conducted both quantitative online surveys and qualitative investigations to understand the prevalence of false positives returned from conventional security tooling and how SOC analysts perceive the quality of an alarm. The authors identified several factors critical to the process of alarm validation, and claimed that integrating these factors into security tooling will enable analysts to make more informed decisions when validating an alarm, and will expedite analyst reactions to alarms. Furthermore, Agyepong et al. [25] conducted several semi-structured interviews of SOC managers from a variety of industries to gain a better understanding of a SOC analyst's core functionalities. The argument provided by the authors is that without a joint agreement on the definition of the SOC analyst's role, it will remain difficult to define the metrics necessary for assessing an analyst's performance.

Active SOC studies go beyond asking questions and additionally ask the participant to perform a cybersecurity-related technical task. For example, in Rosso et al. [26], the authors

[1] https://attack.mitre.org/

constructed a SOC evaluation platform and tested its effectiveness by having students play the role of SOC analysts potentially investigating one of two attack scenarios. The purpose of their platform, SAIBERSOC, is to measure the detection accuracy of various SOC configurations with regard to attack identification and investigation, and attempt to find the best SOC configuration which best facilitated higher detection rates. Additionally, Kersten et al. [27] conducted experiments with a group of tier-1 analysts from an organization's SOC and had a subset of the analysts investigate attacks using their proposed process of alert investigation. The alerts presented were a set of real alerts generated from within their organization's network, and the authors determined that their proposed process resulted in analysts being 2.5 times more likely to build an accurate assessment of the alert.

The user study conducted within this paper is considered an active SOC study, as the participants are actively investigating and analyzing cyber attack scenarios. This work is unique in that we seek to determine the comparative difficulty of using conventional and state-of-the-art methods for performing threat detection.

IV. User Study

This section will detail the structure and execution of a user study performed as part of this research effort[2].

A. Study Overview

This study involved 13 participants, with a background in cybersecurity, completing a sequence of intrusion detection rule creation exercises for attack scenarios of increasing complexity. These participants were recruited through study advertisements made in our institution's cybersecurity club and security-related courses held on campus, where participants who completed the study in full had the chance to win one of five $25 Amazon gift cards. The resulting participant pool contained young adults with a career interest in cybersecurity, who are either university students, recently graduated from university, or have a few years of industry experience post-graduation. The study was performed with participants in a 1-on-1 setting remotely using Zoom, and Qualtrics as the medium for presenting the participants with questions, as well as for collecting results.

This study does not seek to compare and contrast the effectiveness of system provenance graph-based detection and log-based detection methods in general. Instead, this analysis shines a light on the process of attack detection rule creation, and specifically how that process differs between graph-based detection and log-based detection. To accomplish this, given a cyber attack scenario description and the resulting audit logs/graph, we measure the effort involved in creating a detection rule for the given attack scenario. We also measure the success of these rules by running them against attack scenarios emulated in a lab environment. To measure the effort of creating detection rules, participants were timed during their exercises, and the length of the resulting detection rules was also be measured. In addition to these metrics, after each of the exercises have been completed, the participants were also asked questions seeking to compare and contrast their experience when constructing log and graph-based rules.

[2]This user study was approved by Oregon State University's Human Research Protection Program and Institutional Review Board under study application HE-2023-340

B. Study Workflow

Aside from the study description and consent agreement, the study is broken into the following four sections: pre-survey questionnaire, tutorial videos, rule creation exercises, and post-survey questionnaire.

1) Pre-Survey Questionnaire: During the pre-survey questionnaire, participants were asked to state their experience levels for a variety of subject areas. Having an understanding of these concepts would be beneficial when conceptualizing the attack scenario and building detection rules. These concepts include:

- The Linux operating system (e.g., processes, files, common commands)
- Analyzing operating system or application logs (e.g., audit logs, ssh logs, web server logs)
- Regular expressions
- Basics of graph terminology (e.g., nodes, edges, directed graphs)
- Host-based intrusion detection systems (HIDS; e.g., Wazuh)
- Graph databases (e.g., Neo4j)
- Creating cyber threat detection rules (e.g., YARA, SIGMA, Wazuh, Suricata)

For each of these concepts, the participants were asked to rate their experience level on a scale from 1 to 5, where each number represented the following experience levels:

1) No experience/knowledge of this topic
2) Some knowledge of, but no experience with this topic
3) Some knowledge and experience with this topic
4) Extensive knowledge of, but only some experience with this topic
5) Extensive knowledge and experience with this topic

2) Tutorial Videos: The participants were then asked to watch two tutorial videos, both hosted on YouTube and totalling 17 minutes long, which described the process of completing the rule creation exercises provided later in the study. The videos gave participants a crash course on intrusion detection systems, Linux audit logging using Auditd, and how those audit logs can be interpreted as graphs. Next, the videos described how detection rules can be made for detecting audit logs with specific field values, and also how detection rules can be made for audit logs which have been converted into a provenance graph. Finally, the videos described what the upcoming rule creation exercises will look like, what they will include, and how they should be completed. This process was solidifed by going over an example rule creation exercise, including what a correct log and graph-based rule might look like. Also included in the Qualtrics page containing hyperlinks to the tutorial videos is a cheat sheet, which states the expected syntax of the log and graph-based rules, and summarizes the main points of the tutorial videos. The participants were reminded that this cheat sheet would be available on each page which requires them to build a log or graph-based detection rule.

3) Rule Creation Exercises: Each participant was given three rule creation exericses to complete, where each exercise consisted of three separate consecutive pages within the Qualtrics study. The rule creation exercises were given in order of increasing difficulty, where each exercise is defined as being either easy, moderate, or hard. The first page of the exercise contained the attack scenario description, giving the participant an opportunity to understand the attack prior to making the detection rule. The attack scenario description was still available on the second and third pages of each exercise, however these two pages also prompted the participant to construct their detection rule. Whether the participant was prompted to construct a log-based or graph-based detection rule at first was randomized. The participants were also asked to pause prior to submitting their detection rule to Qualtrics, so that the syntax of their detection rule could be checked. If the detection rule contained incorrect syntax, the participants would be asked to correct their mistake. However, when the rule was syntactically correct but was clearly not going to detect the given attack, they were not alerted to the issue or asked to make any changes.

4) Post-Survey Questionnaire: Finally, the participants were asked to answer a series of questions which detailed their experience making detection rules using log and graph-based methods. Participants were also asked to provide insight into their answer as to better understand why they made their choice. Specifically, they were asked the following questions:

1) Between log-based and graph-based rule creation, which do you feel was more difficult?
2) Between log-based and graph-based rule creation, which do you feel you spent more time working on? In other words, which was more time consuming?
3) Between log-based and graph-based rule creation, which do you feel best captured the rules that you were trying to produce?
4) How often did you need to refer to the cheat sheet in order to answer the exercises?
5) Regarding the log-based rule syntax: how confident were you with writing the final exercise's rule, compared to the first exercise's rule?
6) Regarding the graph-based rule syntax: how confident were you with writing the final exercise's rule, compared to the first exercise's rule?

C. Rule Creation Exercises

This section will detail the log and graph-based rule syntax for the user study, the attack selection process, defining difficulty between exercises, and the layout of each rule creation exercise.

1) Exercise Layout: Each rule creation exercise contained within the user study consisted of a sequence of three pages. The first page contained a description, ranging from two to three paragraphs, of a specific attack being performed on a computer. After reading through the attack description on the first page and navigating to the next page, this page and the next page (i.e., the final two pages) of the rule creation exercise entail constructing either a log-based rule, or a graph-based rule, to detect the attack. The order in which the user is presented these final two pages is determined at random. The purpose of creating separate pages is to separate the collection of metrics related to log-based rule creation, and graph-based rule creation. In the page corresponding with the log-based rule, the participant is additionally provided with a section from a Linux Auditd log which correlates with the attack described earlier. For the graph-based rule creation page, this is replaced with a typed and attributed graph, containing details derived from the Linux Auditd logs within the log-based rule creation page, which describes the attack.

2) Difficulty: After the tutorial videos, participants were given a sequence of three rule creation exercises in the following order of difficulty: easy, moderate, then hard. Within each category of difficulty we had three rule creation exercises that were randomly assigned. Qualtrics was configured to ensure that each exercise within the difficulty group got equal exposure to participants. Variance in difficulty was integrated to track how much more/less time and effort it takes the participant to complete exercises as attacks grew in complexity. This insight is important because it identifies how scalable certain detection methodologies are as attack complexity grows. A total of nine rule creation exercises were developed for the user study, with three different exercises for each difficulty category.

Easy rule creation exercises resembled single events performed by a single process and entailed exact field/value matching, where the log-based exercises contained two audit events and the graph-based exercises were described by two nodes connected by a single edge. Moderate rule creation exercises increased the number of processes and system calls involved and utilized simple usage of regular expressions, where log-based exercises had three audit events and the graph-based exercises contained four nodes and three edges. Finally for the hard exercises, additional processes and system events were used along with additional regular expressions of higher complexity. The hard exercises used five audit events for log-based exercises, and graph-based exercises used four nodes and five edges.

3) Detection Rule Syntax: A custom syntax was used when creating detection rules within the exercises — this was done to reduce complexity enough to allow for relatively-quick rule creation, while still making the process similar to what is commonly seen in industry. The goal was to be able to give the participants enough time to complete all three rule creation exercises within one hour, while still completing the tutorial videos and questionnaires. The log-based syntax was based on Wazuh's XML rule formatting converted into CSV format, while the graph rules are primarily based on Neo4j's Cypher graph query langauge.

The log-based syntax utilized during the user study entails comma-separated pairs of fields and values, where each field and value correspond to data observed in the system audit logs. Each log-based rule can contain one or more lines, where each line within the rule targets a particular set of fields and values from a single audit log. For example, Figure 2 depicts a snippet of an audit log file, in addition to a sample log-based threat detection rule. The graph-based syntax used for the user study is closer to Neo4j's Cypher query syntax, compared to the custom log-based syntax's resemblence to Wazuh's XML rule syntax. Cypher is a query language which enables its users to craft typed graphs by defining representations of nodes, edges,

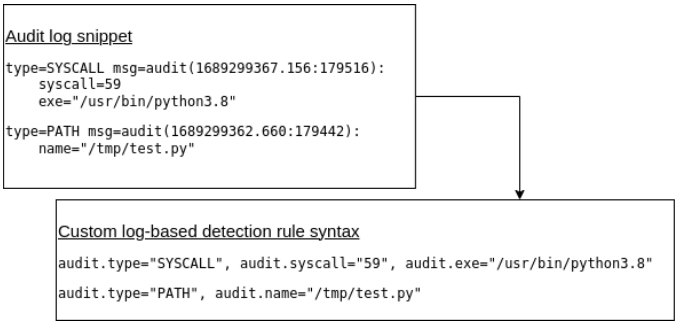

Fig. 2: Example of the Custom Log-Based Detection Rule Syntax

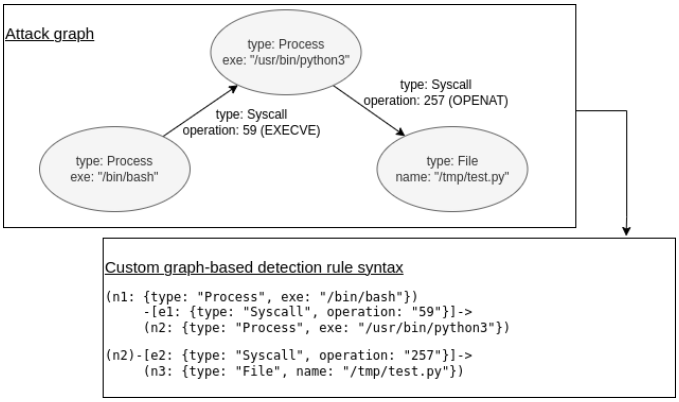

Fig. 3: Example of the Custom Graph-Based Detection Rule Syntax

and their associated attributes, which are then queried against a larger graph to perform exact graph matching. In the custom graph-based syntax, Cypher clauses (such as `MATCH`, `RETURN`, and `WHERE`) were removed so that nodes, edges, and their attributes remained the primary focus to the participants — this is depicted in Figure 3.

4) Attack Selection: There were several factors in play when deciding which attacks to choose for each rule creation exercise. First, the attacks chosen need to fit into one of the three previously-defined difficulty categories. Specifically, each exercise contained within a given difficulty group needed to be similar enough in structure (with regard to the scenario's audit log snippet and attack graph) to justify their placements within each group. The total set of nine attack scenarios also needs to vary in tactics utilized to account for there being several distinct steps which ultimately create an attack killchain. Each difficulty class contains three attack scenarios, and the total set of exercises spans five adversarial tactics described by MITRE's ATT&CK matrix[3]. With regard to relevancy of the attacks, a majority of the attack scenarios selected are mentioned in Mandiant's 2022 APT Trends report[4]. The attacks which were ultimately chosen for the user study are shown in Table I.

[3]https://attack.mitre.org/

TABLE I: User Study Attack Scenario Descriptions

Scenario Difficulty	Scenario ID	Scenario Description	MITRE Tactic
Easy	1.1	Editing .ssh/authorized_keys	Persistence
	1.2	Deleting .bash_history	Defense evasion
	1.3	Netcat usage	Execution
Moderate	2.1	Awk privilege esc.	Privilege esc.
	2.2	.bashrc reverse shell	Persistence
	2.3	Vim defense evasion	Defense evasion
Hard	3.1	PHP webshell	Persistence
	3.2	Download/Execute Python script	Execution
	3.3	Download/Install systemd module/exe	Persistence

V. Analysis of Results

This section analyzes the results of the user study from two perspectives, rule creation difficulty and the resulting rule's efficacy in detecting threats.

A. Comparative Analysis of Difficulty

This analysis seeks to compare the time and effort involved with creating log and graph-based rules, while also integrating context from both the pre and post-survey questionnaires.

1) Pre-Survey Questionnaire: To gain a better understanding of the participants prior experience using the the pre-survey questionnaire, scores were assigned to each of the possible answers within each subject area. Below is a list of pre-survey questionnaire responses, followed by their score.

- "No knowledge or experience" = 0
- "Some knowledge but not experience" = 1
- "Some knowledge and some experience" = 2
- "Extensive knowledge and some experience" = 3
- "Extensive knowledge and experience" = 4

The average summed experience per participant was 16.4, and dividing the average experience by the seven subject areas gives us a general experience score (across all subjects) of roughly 2.35. So on average, participants were between somewhat knowledgeable and experienced in each subject, and being extensively knowledgeable but only having some experience in the area. Participants seemed to be most familiar with the Linux OS and analyzing OS/application log files (average score of 2.9 and 3 respectively), while being the least comfortable with graph databases (average score of 1.2). This data indicates that the participants were more comfortable with subject areas relating to general computer science topics — regular expressions and graph terminology — as opposed to cybersecurity-focused topics such as detection rule creation and HIDS. These results associated with the pre-survey questionnaire can be found in Table II.

2) Rule Creation Time: We notice distinct trends which separate the learning curves associated with log and graph-based rule creation, the details of which are found in Table III. First, the average and median log-based rule creation times decrease as difficulty moves from easy to moderate, and increase as difficulty goes from moderate to hard. However, the opposite is observed for average and median graph-based rule

TABLE II: Pre-Survey Questionnaire Statistics

Pre-Survey Questionnaire Topic	Min	Max	Mean	Median
Linux OS	1	4	2.9	3.5
OS/Application log analysis	1	4	3	5
Regular expressions	1	4	2.2	2
Graph basics	1	4	2.7	3
HIDS	1	4	2.1	2
Graph databases	0	3	1.2	1
Detection rule creation	1	4	2.1	2

TABLE III: Log and Graph-Based Rule Creation Time Statistics (Time Given in Seconds)

	Scenario Category	Min	Max	Mean	Median
Log-Based	Easy	202.21	566.12	308.14	273.39
	Moderate	137.82	468.89	257.86	230.76
	Hard	136.37	403.45	274.16	285.19
Graph-Based	Easy	160.37	661.34	384.84	364.22
	Moderate	299.21	791.4	458.96	435.89
	Hard	293.51	569.79	422.63	426.40

creation times. We hypothesize that this is due to participants needing to implement additional graph-specific rule creation concepts (such as node and edge aliasing, and multi-edge connections) as difficulty increases. Although as the participants gain experience in creating log and graph-based rules, they require less time to implement previously-seen concepts. We additionally observe that, on average, graph-based rule creation takes noticeably longer than log-based rule creation. During the easy exercises, the difference in log and graph-based rule creation time is only about 80 seconds, although this difference expands to over 140 seconds when completing hard exercises. Within the context of these exercises, it appears that participants will require an increasing amount of time to build graph-based rules as attack scenario complexity rises, compared to log-based rules.

3) Rule Length: Regarding rule length, during easy rule-creation exercises, the participants wrote shorter graph-based rules (126 characters) on average, than log-based rules (152 characters). However, this changes when moving to moderate difficulty exercises, where participants wrote graph-based rules which were 280 characters long on average, compared to 215 characters for log-based. The same trend is seen with hard rule creation exercises, where although the gap between the two is shorter, graph-based detection rules are still 339 characters long on average, compared to 324 for log-based rules. This indicates that within the context of these exercises, although graph-based rules take less characters to write at first, for more complex scenarios, they do require more characters to write than log-based rules. The details surrounding log and graph-based rule length can be found in Table IV.

4) Post-Survey Questionnaire: The questions from the post-survey questionnaire can be found in subsection IV-B4, while the results of this questionnaire can be found in Table V. From the first two questions, although most believed that the log and graph-based exercises were equal in difficulty, a majority of participants felt that they spent more time on graph-based rule creation. Participants stated that the information needed to complete each task is present, and that the remaining

TABLE IV: Log and Graph-Based Rule Length Statistics (Length Given by Number of Characters)

	Scenario Category	Min	Max	Mean	Median
Log-Based	Easy	117	174	152.61	154.5
	Moderate	185	231	215.92	219
	Hard	274	348	324.46	333
Graph-Based	Easy	105	143	126.15	126.5
	Moderate	253	299	280.69	283
	Hard	293	368	339.15	343

TABLE V: Post-Survey Questionnaire Statistics

Post-Survey Question #	Response	# Answers
1 (Difficulty)	Graph-based slightly more difficult	3
	Equally difficult	7
	Log-based slightly more difficult	3
2 (Time Spent)	More time spent on log-based	1
	Time spent equally	2
	More time spent on graph-based	10
3 (Usability)	Log-based rules captured better	1
	Both equally captured the rule	2
	Graph-based captured better	10
4 (Cheat Sheet)	Only a few times	10
	Frequently	2
	1 or more times per exercise	1
5 (Graph Confidence)	Somewhat less confident	2
	Equally confident	3
	Somewhat more confident	3
	Much more confident	5
6 (Log Confidence)	Much less confident	1
	Somewhat less confident	1
	Equally confident	5
	Somewhat more confident	1
	Much more confident	5

task is simply to transpose the data into the required form. One participant noted that the graph-based rules "makes more sense," however that the syntax was much more "painful." A majority of participants noted that graph-based rules better captured what they were writing, with one participant noting that it's "... easier to visualize what you're producing, instead of just looking at the log itself." Regarding cheat sheet usage, participants felt that once they began producing both types of rules, they only needed to look back when they needed to implement a new concept they hadn't done previously (such as node aliases). For the final two questions regarding rule creation confidence, it appears that a roughly even amount of participants felt more or less confident by the time they finished their final graph-based exercise, while a majority of participants noted an equivalent or increase in confidence for log-based exercises.

B. Comparative Analysis of Efficacy

The goal of this analysis is to qualitatively evaluate the performance of the detection rules created by the participants in the user study.

1) Overview: To evaluate these rules, system data was generated which represents the attacks shown in each scenario. For log-based data, this came in the form of audit logs, while SPADE [28] was utilized to generate audit graph data within

TABLE VI: Detection Accuracy of Log and Graph-based Participant Rules

	Participant Rules (Divided by Rule Creation Exercise)								
	Easy			Moderate			Hard		
	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3
Attacks Detected Using Log-based Rules	4	5	3	6	3	3	6	4	3
Attacks Detected Using Graph-based Rules	4	5	4	6	3	3	4	4	3
Total Number of Exercises	4	5	4	6	4	3	6	4	3

a Neo4j graph database. Next, each of the rules created by the participants was transformed into a Wazuh (log-based) and Neo4j Cypher (graph-based) rules. The process of transforming the participant rules into rules that could be utilized by Wazuh and Neo4j is detailed in Appendix A, while the difficulties associated with this process are described in Appendix B. Finally, each of the rules was ran against their associated underlying dataset to determine whether or not the rules raised an alert.

2) Synthetic Dataset Generation: To evaluate the rules produced by the participants, each attack scenario was emulated and the resulting logs were collected. This was accomplished by running each attack on an isolated endpoint within our lab environment while running Linux Auditd. High granularity Auditd configurations were utilized, including file and network I/O syscall tracking, in an effort to collect as much details as possible during the attack emulation. The dataset collected included the attack behavior being emulated, in addition to benign background system activity that took place as the attack scenario was emulated.

3) Detection Accuracy: To infer detection accuracy, we find the ratio of participants (per attack scenario) who were able to make a detection rule which produced a true positive alert for the malicious behavior within the log/graph. The missed detections in this portion of the analysis will be considered false negatives in the subsequent subsection. The results of this analysis exist in Table VI.

In a majority of cases the participants were able to create accurate rules to detect the attack behavior for both log and graph-based scenarios. When participants' rules fail to detect a given attack, it's often due to simple typing mistakes. Common occurrences that were found include using incorrect filepaths, and incorrectly defining a regular expression. In total, log-based detection was able to achieve one more detection than graph-based — out of the 39 total exercises among participants, it's clear that both methods were both capable of detecting the attacks thrown at them.

4) False Positives and False Negatives: The number of false positives identified are shown in Table VII, while the false negative count can be derived from the missed detections seen in Table VI.

We observe that graph-based rules are a clear winner in terms of false positives when compared with log-based rules. This is partially due to the Wazuh's design which does not natively support rule correlation, similar to what aliasing in Neo4j's Cypher language accomplishes. However we also hypothesize that this is due to graph queries containing all entities

TABLE VII: Average False Positive Count of Log and Graph-based Participant Rules

	Participant Rules (Divided by Rule Creation Exercise)								
	Easy			Moderate			Hard		
	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3
Log-based Rules	80	5.8	0.75	4	44	2298.3	369	34	1479
Graph-based Rules	0	0	0	0	59	0	0	0	0

and events being mentioned in the query, as opposed to log-based rules which independently alert on specific occurrences of entities and events. That being said, it's clear that there's still the potential to create graph-based rules which are specific enough but end up producing false positives (see the 59 for 2.2 in Table VII) Additionally, we observe that false positives increase for log-based rule creation as difficulty increases — this is due to more sub-rules being involved within the parent rules as difficulty increases, which creates additional independent Wazuh rules and increases the potential for false alerts. Regarding false negatives, comparing the results derived from this evaluation gives us that both log and graph-based detection produces low numbers of false negatives, with log-based only having a total of two false negatives and graph-based with three.

VI. Conclusions & Future Work

This work analyzed results from a user study that measured the variation of difficulty faced by security analysts when creating graph-based and log-based threat detection rules. The subsequent analysis consisted of qualitatively assessing the effort involved in the rule creation process and evaluating rules derived from the user study. Through this analysis, we observed that the creation process for graph-based rules has additional overhead but provides higher levels of insight and interpretability. Additionally, the resulting rulesets in a rule-based PIDS deployment produce equivalently accurate detection results, but naturally provide more contextual information in addition to much fewer false positives.

Due to the scale and setting of this research effort, there are a few notable limitations which define areas for future work. First, this study analyzed the comparative difficulty between log-based and graph-based detection rule creation, so the results collected will not make generalized assumptions regarding performance or efficiency between log and graph-based detection methods. Larger-scale studies analyzing the difficulty from both qualitative and quantiative points of view are left to future work. Additionally, to keep the user study concise in nature, the attacks chosen are noticeably more compact than other scenarios that could have been chosen, and this study did not strenuously evaluate the participants' ability to perform analysis and correlation on larger sets of logs/graphs. We additionally leave longitudinal studies involving the wider selection of conventional security tooling to compare against graph-based methods for future work.

Acknowledgment

The authors would like to thank Rebecca Joshua of Oregon State University for her support of this research effort.

References

[1] Checkpoint, *Check Point Research Reports a 38% Increase in 2022 Global Cyberattacks*, 2023. [Online]. Available: https://blog.checkpoint.com/2023/01/05/38-increase-in-2022-global-cyberattacks/.

[2] CrowdStrike, "2023 Global Threat Report," Tech. Rep., 2023. [Online]. Available: https://www.crowdstrike.com/global-threat-report/.

[3] C bibinitperiodISA (CISA), *Advanced Persistent Threats and Nation-State Actors*. [Online]. Available: https://www.cisa.gov/topics/cyber-threats-and-advisories/advanced-persistent-threats-and-nation-state-actors.

[4] Mandiant, "Mandiant M-Trends 2022 Special Report," Mandiant, Tech. Rep., 2022. [Online]. Available: https://www.mandiant.com/media/15671.

[5] G Karantzas and C Patsakis, "An empirical assessment of endpoint detection and response systems against advanced persistent threats attack vectors," *Journal of Cybersecurity and Privacy*, vol. 1, no. 3, pp. 387–421, 2021, Publisher: MDPI. DOI: 10.3390/jcp1030021.

[6] TM Corporation, *DLL Side-Loading*, 2020. [Online]. Available: https://attack.mitre.org/techniques/T1574/002/.

[7] X Han, *Using Provenance for Security and Interpretability*, 2018. [Online]. Available: https://conferences.inf.ed.ac.uk/EuroDW2018/papers/eurodw18-Han.pdf.

[8] A Bates and WU Hassan, "Can Data Provenance Put an End to the Data Breach?" *IEEE Security & Privacy*, vol. 17, no. 4, pp. 88–93, 2019. DOI: 10.1109/MSEC.2019.2913693.

[9] Z Li, QA 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, ISSN: 0167-4048. DOI: 10.1016/j.cose.2021.102282.

[10] M Zipperle, F Gottwalt, E Chang, and T Dillon, "Provenance-Based Intrusion Detection Systems: A Survey," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 0360–0300, 2022, Place: New York, NY, USA Publisher: Association for Computing Machinery, ISSN: 0360-0300. DOI: 10.1145/3539605.

[11] F Dong, S Li, P Jiang, D Li, H Wang, L Huang, X Xiao, J Chen, X Luo, Y Guo, and X Chen, "Are we there yet? An Industrial Viewpoint on Provenance-based Endpoint Detection and Response Tools," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, _eprint: 2307.08349, Association for Computing Machinery, 2023, pp. 2396–2410. DOI: 10.1145/3576915.3616580.

[12] A Gupta, "Data Provenance," in *Encyclopedia of Database Systems*, L LIU and MT ÖZSU, Eds., Springer US, 2009, pp. 608–608, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_1305. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_1305.

[13] K Scarfone and P Mell, "NIST SP 800-94: Guide to Intrusion Detection and Prevention Systems (IDPS)," National Institute of Standards and Technology (NIST), NIST Special Publication, 2007, p. 127. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-94.

[14] M Liu, Z Xue, X Xu, C Zhong, and J Chen, "Host-Based Intrusion Detection System with System Calls: Review and Future Trends," *ACM Comput. Surv.*, vol. 51, no. 5, 2018, Place: New York, NY, USA Publisher: Association for Computing Machinery, ISSN: 0360-0300. DOI: 10.1145/3214304. [Online]. Available: https://doi.org/10.1145/3214304.

[15] F Research, "The 2020 state of security operations," Tech. Rep., 2020. [Online]. Available: https://www.paloguard.com/datasheets/forrester-the-2020-state-of-security.pdf.

[16] A Wolf, *Cybersecurity alert fatigue*. [Online]. Available: https://arcticwolf.com/cybersecurity-alert-fatigue/.

[17] J Zeng, ZL Chua, Y Chen, K Ji, Z Liang, and J Mao, "WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics," in *Network and Distributed Systems Security (NDSS) Symposium 2021*, NDSS, 2021. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2021-549-paper.pdf.

[18] MN Hossain, SM Milajerdi, J Wang, B Eshete, R Gjomemo, R Sekar, S Stoller, and V Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *26th USENIX Security Symposium (USENIX Security 17)*, USENIX Association, 2017, pp. 487–504, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hossain.

[19] K Pei, Z Gu, B Saltaformaggio, S Ma, F Wang, Z Zhang, L Si, X Zhang, and D Xu, "HERCULE: Attack Story Reconstruction via Community Discovery on Correlated Log Graph," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC '16, Association for Computing Machinery, 2016, pp. 583–595, ISBN: 978-1-4503-4771-6. DOI: 10.1145/2991079.2991122.

[20] WU Hassan, S Guo, D Li, Z Chen, K Jee, Z Li, and A Bates, "NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage," in *Network and Distributed Systems Security Symposium*, NDSS, 2019. [Online]. Available: https://par.nsf.gov/biblio/10085663.

[21] WU Hassan, A Bates, and D Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems," in *2020 IEEE Symposium on Security and Privacy (SP)*, ISSN: 2375-1207, IEEE, 2020, pp. 1172–1189. DOI: 10.1109/SP40000.2020.00096.

[22] Y Liu, M Zhang, D Li, K Jee, Z Li, Z Wu, J Rhee, and P Mittal, "Towards a Timely Causality Analysis for Enterprise Security.," in *Network and Distributed Systems Security Symposium*, NDSS, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07B-3_Liu_paper.pdf.

[23] FB Kokulu, A Soneji, T Bao, Y Shoshitaishvili, Z Zhao, A Doupé, and GJ Ahn, "Matched and Mismatched SOCs: A Qualitative Study on Security Operations Center Issues," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, Association for Computing Machinery, 2019, pp. 1955–1970, ISBN: 978-1-4503-6747-9. DOI: 10.1145/3319535.3354239.

[24] BA 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)*, USENIX Association, 2022, pp. 2783–2800, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/system/files/sec22-alahmadi.pdf.

[25] E Agyepong, Y Cherdantseva, P Reinecke, and P Burnap, "Towards a Framework for Measuring the Performance of a Security Operations Center Analyst," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, IEEE, 2020, pp. 1–8. DOI: 10.1109/CyberSecurity49315.2020.9138872.

[26] M Rosso, M Campobasso, G Gankhuyag, and L Allodi, "SAIBERSOC: A Methodology and Tool for Experimenting with Security Operation Centers," *Digital Threats*, vol. 3, no. 2, 2022, Place: New York, NY, USA Publisher: Association for Computing Machinery. DOI: 10.1145/3491266.

[27] L Kersten, T Mulders, E Zambon, C Snijders, and L Allodi, "'Give Me Structure': Synthesis and Evaluation of a (Network) Threat Analysis Process Supporting Tier 1 Investigations in a Security Operation Center," in *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, USENIX Association, 2023, pp. 97–111, ISBN: 978-1-939133-36-6. [Online]. Available: https://www.usenix.org/conference/soups2023/presentation/kersten.

[28] A Gehani and D Tariq, "SPADE: Support for Provenance Auditing in Distributed Environments," in *Middleware 2012*, P Narasimhan and P Triantafillou, Eds., Springer Berlin Heidelberg, 2012, pp. 101–120, ISBN: 978-3-642-35170-9. DOI: 10.1007/978-3-642-35170-9_6.

APPENDIX A
EVALUATION METHODS

To compare the performance of the log and graph-based rules against the synthetic dataset, the participant-made rules must be translated into a standard rule format to be compared against the audit logs and system provenance graphs, respectively. To accomplish this, custom scripts were written to translate each of the log-based rules into an XML format which can be utilized by Wazuh to perform rule-based detection. Additionally, each graph-based rules had slight modifications made to it in order for them to work as Neo4j Cypher queries.

To evaluate log-based rules against an audit log, Wazuh's `wazuh-logtest` tool was used to compare each entry in the attack scenario's audit log to the participant-made rules. Each line within a participant's log-based rule would be converted into a single XML-based Wazuh rule. These individual rules would be appended together and evaluated against the target attack scenario's audit log using `wazuh-logtest`, which raises alerts to the user when audit log activity matches a rule. For graph-based evaluations, the attack scenario's audit logs were converted into a system provenance graph using SPADE, which were then subsequently fed into a Neo4j database. Once the graphs were indexed into the database, the queries derived from the participant-made rules from the user study were ran against the database; results returned from these queries were considered alerts.

APPENDIX B
TRANSLATING FROM CUSTOM SYNTAX TO WAZUH/CYPHER

Translating from the custom log-based syntax to Wazuh syntax was straightforward, however several ad-hoc changes were required to be made when translating the custom graph query syntax from the rule creation exercises, to Neo4j Cypher queries. This was due to issues encountered with SPADE when creating the system provenance graph, including reversed edges and discrepancies between expected behavior and observed behavior. When translating between the rule creation exercise graph queries and Neo4j Cypher queries, we permitted using bidirectional edges as many edges defined by SPADE were reversed when compared to the actual system behavior. In other cases, there were mistakes made during the synthesis of the dataset which complicated downstream analysis tasks. For instance, in log and graph-based exercises 1.2 and 2.2, the target file related to the attack scenario did not match the full path shown in the rule creation exercises — as a result, the target file path in the query needed to be adjusted when applicable. Additionally in exercise 2.1, new versions of the `sudo` executable break the attack graph pattern provided to the participants, so the target of the forking operation was instead moved to the `gawk` executable. Finally, in the hard exercises, the source of the disk file writes needed to be switched from the socket to the parent process for graph-based rules.