Degree Project

UNIVERSITY
OF SKÖVDE

Segmentation and Dynamic Expansion of IDS Rulesets

Master Degree Project in Informatics with a specialization in Privacy, Information and Cyber Security

Second Cycle 30 credits

Spring term 2024

Student: Theertharaja Bannikere Eshwarappa

Supervisor: Sten F Andler

Industrial Supervisor: Yousef Hashem, Ericsson AB

Examiner: Ali Padyab

Abstract

This research explores an innovative approach to managing extensive rulesets in Host Intrusion Detection Systems (HIDS) through segmentation and dynamic expansion. Drawing upon the MITRE ATT&CK framework, the methodology categorizes rulesets into initial detection, choke point detection, and advanced detection, streamlines threat detection, and optimizes resource utilization. The segmentation allows for targeted detection of potential threats, while dynamic expansion enables the addition of advanced detection rules based on attacker actions. The study evaluates the effectiveness of this approach in reducing performance overhead and improving threat detection capabilities. Test cases validate the approach for detecting multi-stage attacks and optimizing system performance. Results indicate that while the segmentation and dynamic expansion technique offers structured threat detection, challenges such as missed detections and complexity in rule management exist. Future research directions include refining segmentation processes and enhancing rule categorization logic. Overall, this research contributes to the advancement of HIDS methodologies and underscores the importance of ongoing refinement and validation in cybersecurity strategies.

Keywords: intrusion detection systems, rule management, MITRE ATT&CK framework, segmentation, dynamic expansion, system performance

Contents

List of Figures

1. Introduction

In a world buzzing with technology, there's a silent force driving our modern society: embedded systems. These unassuming yet powerful entities silently orchestrate countless operations behind the scenes, serving as the foundation of our interconnected world. Encompassing everything from smart home devices to industrial automation, they ensure the seamless transmission of data through a delicate balance of hardware and software. From the familiar presence of household Wi-Fi routers and street cameras to car navigation systems, even the precision of personal gadgets like fitness trackers and drones, embedded systems quietly shape our lives, keeping us at the forefront of the digital revolution.

Key trends include the advanced integration of technology in areas such as vehicle safety, security systems, and IoT applications, often incorporating computer vision and AI. For instance, in 2023, the number of connected IoT devices surged to 16.7 billion, highlighting a substantial increase in internet-connected devices (Cole 2024). This underscores the exponential growth of embedded system usage and how it has become an integral part of daily life. Despite their transformative potential, the rapid growth of embedded devices also raises significant security concerns. These devices often lack advanced security measures due to limited hardware resources like CPU capacity, memory, and power (Bures et al. 2021). As a result, modern security mechanisms are frequently absent, making embedded systems vulnerable to attacks. Moreover, the design priorities of embedded systems prioritize efficiency and compactness over robust security measures (Sen, Koo, and Bagchi 2018). Additionally, manufacturers often need to pay more attention to providing security updates for affordable products, and users frequently overlook firmware updates (Bures et al. 2021). Consequently, the large number of embedded devices makes them easy targets for potential attacks. These significant security issues underscore the importance of addressing concerns for embedded systems to safeguard against potential risks in our ever-expanding interconnected world.

1.1. Problem Description

To address the security issue of the embedded system device and detect the potential threat in the first place, the Intrusion detection system (IDS) serves as the key protective tool. It provides the coverage to detect the potential intruder in the network or at the host and, accordingly, notify the system administrator upon detecting the security breaches. There are two main types of IDS: Network-based IDS (NIDS) and Host-based IDS (HIDS). NIDS monitors network traffic for malicious activities by connecting to one or more network segments. Meanwhile, HIDS is installed directly on a computer device to monitor for malicious activities occurring within the system. Unlike NIDS, HIDS doesn't solely focus on network traffic but also examines system calls, running processes, file-system changes, interprocess communication, and application logs (Zarpelão et al. 2017).

HIDS are highly effective in protecting host security by directly monitoring suspicious process behaviors on the host itself. Although some of its functions can be centralized, it is typically deployed directly on the target host. To identify threats, there are two main methods: one is based on predefined patterns in its ruleset, and the other involves using anomaly-based detection to flag deviations from the normal behavior of the system, depending on either the system rules or the models (Zarpelão et al. 2017). Both strategies utilize rulesets to trigger alerts upon detecting any suspicious activities. The rulesets are typically updated when an incident is discovered in the system. In most instances, rules are kept unchanged to maintain coverage for historical security events that can happen in the future, with occasional deletions and purges occurring in specific rule sets (Vermeer, Eeten, and Gañán 2022).

Incorporating HIDS into embedded devices poses challenges due to their limited computational resources and strict operational demands (Zarpelão et al. 2017). The utilization of rulesets in intrusion detection can significantly impact a host's computational capabilities, leading to increased processing demands and memory usage when matching against a large ruleset for a single event (Vermeer, Eeten, and Gañán 2022). In addition, the research by Ji et al. (2015) indicates that detection methods based on abnormal behaviors, such as intercepting system calls and performing complex calculations, consume substantial CPU and memory resources, particularly noticeable when multiple programs are running, causing increased host overhead, and affecting normal usage patterns. This additional resource usage by the system or program during its regular tasks contributes significantly to performance overhead (Ji et al. 2015). Therefore, minimizing the performance overhead of the IDS is essential for maintaining normal system operation.

Embedded systems with limited capacity may struggle to manage extensive rulesets alongside other operations, often requiring smaller, optimized rulesets for efficient intrusion detection without overwhelming the system. Research by Vermeer, Eeten, and Gañán (2022) reveals that a mere 0.5% of all rules contribute to over 80% of alerts and incidents, with only 1.2% of alerts warranting further investigation. This research highlights how a small subset of rules can have a significant impact on triggering alerts, stressing the importance of careful investigation into potential security incidents. Additionally, the concentrated generation of alerts could put a strain on host resources, underscoring the necessity for efficient rule management to ensure optimal system performance.

1.2. Research Aim and Research Questions

The objective of this research is to tackle the obstacles encountered in optimizing HIDS within embedded systems. The primary aim is to explore innovative strategies for effectively managing extensive rulesets utilized in intrusion detection while considering the limitations imposed by the computational resources of embedded systems. The overarching goal is to reduce the performance overhead without compromising security measures. Ultimately, this aims to answer questions related to strategically managing rulesets to ensure comprehensive threat coverage while maintaining the efficiency of intrusion detection within embedded environments.

1. How can segmentation and dynamic expansion techniques within HIDS effectively manage extensive and diverse rulesets?
2. How can performance overhead be minimized on nodes during normal run-time operations?

1.3. Delimitation

This research focuses exclusively on HIDS using signature-based detection methods, specifically designed for various Linux distributions. Tasks related to responding to detections, such as taking action against detections, are outside the scope of this study, and the removal of any added segments is also beyond the study's scope.

1.4. Thesis Structure

The remaining part of the study is structured as follows. In Chapter 2, the main components of the study are introduced alongside a comprehensive review of related work, providing essential background context for the research. Chapter 3 details the research methodology, outlining the data collection and analysis methods employed throughout the study. Moving forward, Chapter 4 delves into the experimental results obtained from the research, offering insights into the outcomes of various tests and analyses. Following this, Chapter 5 discusses the interpretation of the results, assesses the validity and reliability of the experiments, evaluates the research contributions, addresses encountered limitations, and proposes potential improvements. Lastly, Chapter 6 concludes the study by summarizing key findings, reflecting on the overall significance of the research within the field, and suggesting directions for future work.

2. Background

This chapter provides background information relevant to the research topic and goals. Its purpose is to provide essential context for understanding the thesis. Additionally, it includes a systematic literature review (SLR) and a list of research articles that have been examined.

2.1. Rule Management

A ruleset consists of predefined rules or signatures used to detect specific patterns or behaviors associated with known security threats or attacks (Zarpelão et al. 2017). These rules are loaded into the system to identify malicious or unauthorized actions and trigger alerts. Research indicates that rules-based systems are commonly used in IDS (Zarpelão et al. 2017). In evaluation studies conducted by Thongkanchorn, Ngamsuriyaroj, and Visoottiviseth (2013), three intrusion detection systems underwent testing across various attack types, ruleset sizes, and network traffic rates. The Transmission Control Protocol (TCP), a fundamental protocol governing internet communication, was a key focus. TCP traffic, known for its reliability and ordered delivery, generally experiences minimal packet loss and low CPU usage (Wikipedia 2019). However, as TCP traffic rates escalate, increased traffic rates correlate with heightened CPU usage, packet loss, and alert generation. Additionally, employing more rulesets is associated with an increase in generated alerts. Notably, the study does not delve into the root causes of these performance limitations, such as potential inefficiencies within the rules themselves. Furthermore, in the context of Snort IDS performance, Sen (2006) explored the relationship between varying bandwidth and ruleset size. The findings indicated that as bandwidth increases, the ruleset size must decrease to maintain Snort's error rate below a certain threshold.

Similarly, in the study conducted by Asad and Gashi (2018), an analysis of the evolution and similarities among four different rulesets—Snort's official Community, Registered, and Subscribed rulesets, along with the ET Open Suricata ruleset—reveals a mere 1% overlap in rules, indicating limited similarity. Consequently, this minimal similarity results in limited overlap in alert-triggering behaviour between the rulesets from ET and Snort. Therefore, the results suggest a potential impact on the system, implying that employing additional rulesets could increase alert volumes, thereby affecting system resources such as CPU usage and potentially influencing system performance. Moreover, the research conducted by Vermeer, Eeten, and Gañán (2022) highlights an intriguing observation regarding rule updates within IDS environments. It reveals that most rule updates are minor activities, with most rules remaining unmodified and only a fraction being deleted, except for periodic surges in some sets. These updates are typically prompted by shifts in the threat landscape and endeavours to decrease the number of alerts by creating more targeted rules. In certain instances, the failure to examine rules before system activation to ensure optimal protection was highlighted in a comprehensive study conducted by Turner et al. (2016). Among the

118 rulesets stored in the rules folder, a substantial majority (87%) were found to be initially deactivated. While this might reduce the performance overhead on the system, it fails to prepare it to detect and respond to threats adequately. This can result in increased vulnerability and potentially compromise security. Additionally, the study by Vermeer, Eeten, and Gañán (2022) suggests common practices for managing rules. These practices include employing multiple rulesets simultaneously to address gaps in threat coverage, creating proprietary rules tailored to specific client threats and updating them frequently, and reducing false positive incidents by updating triggering rules associated with alerts. While these practices aim to enhance threat coverage and reduce false positives, their specific impact on system performance is uncertain due to a lack of comprehensive studies. Building on the importance of effective rule management, while the main emphasis has been on reducing the number of rules for enhancing performance, there is a growing recognition of the necessity to expand these rule collections for comprehensive threat coverage, including the detection of zero-day threats (Soliman, Sobh, and Bahaa-Eldin 2021). Despite the anomaly based IDS's ability to detect novel attacks, their limited deployment in the industry is primarily due to high false positive rates and the challenge of interpreting trained models (Liu, Hagenmeyer, and Keller 2021). This highlights the importance of striking a balance between rule reduction for performance optimization and the inclusion of additional rules for broader threat detection capabilities.

2.2. Dynamic Expansion Rules

The concept of dynamically expanding rulesets involves creating or adjusting rules in real time. This enables the IDS to quickly adapt to evolving threats without sacrificing efficiency (Liu, Hagenmeyer, and Keller 2021). These techniques, such as automated rule generation based on machine learning algorithms, play a vital role in detecting rapidly evolving threats in the cybersecurity landscape and addressing emerging vulnerabilities or attack vectors (Soliman, Sobh, and Bahaa-Eldin 2021). However, implementing dynamic expansion of IDS rulesets requires careful consideration, particularly recognizing that alterations in the ruleset can have a significant impact on system performance (White, Fitzsimmons, and Matthews 2013).

Liu, Hagenmeyer, and Keller (2021) conducted a survey that outlined several rule-learning techniques used for rule generation. These techniques include Association Learning, Rough Set Theory, Genetic Network Programming, Learning Classifier Systems, Particle Swarm Optimization Algorithm, Sequential Pattern Mining, Markov Chain Decision Rule, Decision Tree, Inductive Logic Programming, Bayesian Network, Radial Basis Function Network, and Episode Mining.

2.3. Intrusion Detection in Embedded Systems

To effectively address cybersecurity threats, it is crucial to incorporate IDS into embedded systems. Typically, these systems gather and preprocess data before analysing it for potential intrusions, subsequently alerting users upon detecting threats. However, enhancing IDS functionality within embedded systems, such as IoT devices, remains

a pressing concern in modern cybersecurity (Zarpelão et al. 2017). Adapting conventional IDS approaches to IoT environments is complicated by limitations in device resources and the unique protocol stacks employed (Zarpelão et al. 2017).

In response to the challenge of energy efficiency and event processing, especially pertinent for resource-constrained devices, a study by Arshad et al. (2019) introduced the collaborative IDS (COLIDE) framework with the Contiki operating system (OS) to achieve effectiveness within an IoT system. They conducted a comprehensive assessment of this framework to identify any potential performance compromises. The study's results illustrate the framework's effectiveness in reducing energy and processing overheads, thereby facilitating collaborative intrusion detection in IoT systems. Particularly noteworthy is the significant decrease in overall energy consumption and memory overheads, highlighting its efficacy in IoT devices with limited resources.

The performance of IDS within embedded system environments is significantly influenced by several key factors. Understanding and addressing these factors is imperative for optimizing the functionality and efficiency of intrusion detection in IoT systems. Research conducted by Ji et al. (2015) highlights three primary factors influencing the overhead of approaches: interception mechanism on the host, type and volume of intercepted system calls, and correlation engine. The suggestions provided by studies Ji et al. (2015) for enhancing the correlation engine include reducing time and space complexity in correlation methods, optimizing calculation modules, avoiding exhaustive enumeration during database matching, and minimizing unrelated matches by improving indexes and buffer pools (Ji et al. 2015). Moreover, efforts to reduce real-time computing should be explored, such as increasing offline preprocessing to alleviate online detection overhead.

2.4. Research Gap

The current body of literature primarily concentrates on assessing IDS tools and their performance, often neglecting the specific impact of ruleset management on system-level efficiency. While some studies touch upon influential factors, there is a clear absence of research dedicated to defining optimal rulesets that balance performance and threat coverage comprehensively. Moreover, existing research tends to focus on comparing ruleset size and its impact on performance, overlooking a thorough examination of ruleset volume versus effectiveness, particularly concerning devices with limited resources. Additionally, findings suggest that larger rulesets result in increased system overhead, underscoring the necessity for studies aimed at reducing rulesets while maintaining effective threat coverage. Addressing these gaps would offer valuable insights into optimizing ruleset management to reduce the performance overhead on the node in IDS environments.

This study explores the complex domain of managing extensive rulesets used in intrusion detection, seeking to find a harmonious balance between robust security measures and the inherent computational resource constraints found in embedded systems.

2.5. Systematic Literature Review

The systematic literature review (SLR) conducted in this study draws upon the framework outlined in 'A Guide to Conducting a Standalone Systematic Literature Review' by (Okoli 2015). This guide provides a structured approach for conducting the literature review, encompassing the following key steps:

2.5.1 Purpose

The purpose of this research is to address the challenges faced in optimizing HIDS within embedded mobile network equipment. The primary objective is to explore innovative strategies for efficiently managing extensive rulesets used in intrusion detection, all while considering the constraints imposed by the computational resources of embedded systems. The ultimate goal is to reduce the performance overhead while maintaining stringent security measures.

2.5.2 Protocol

The initial plan for conducting Systematic Literature Review (SLR) has been carefully crafted, drawing insights from Okoli (2015) literature review guide. This structured approach outlines specific research questions and objectives in clear terms. The protocol provides detailed guidelines for the systematic methodology, including criteria for selecting and excluding studies. It also establishes transparent screening criteria and justifications for study exclusions. Additionally, the protocol specifies the strategy for the literature search, detailing the databases to be used and the search terms to ensure thoroughness. In essence, this protocol serves as a transparent framework, ensuring a methodologically robust and replicable SLR process that will effectively guide team's training.

2.5.3 Screening Procedure

The review exclusively targets studies that align with its specific research question, as outlined by Dawson and Ferdig (2006). Consequently, only studies conducted in English will be considered. Additionally, selected studies must meet the following criteria: relevance to keywords and theme, publication within the last 20 years with priority given to newer publications, peer-reviewed status, and publication in a recognized journal. These criteria are essential for maintaining the integrity and relevance of the review process.

Search terms

This study aims to explore the utilization of rule sets in intrusion detection systems, particularly focusing on embedded systems. The goal is to find a balance between ensuring system security and optimizing computational resources. To achieve this objective, specific search terms related to intrusion detection have been carefully selected. During the search process, encountered some texts that may not have been strictly academic or peer-reviewed. However, these texts were included due to their relevance to the thesis, while maintaining professionalism in selecting additional material not found

in the literature review. Additionally, given the research's specific focus on assessing the impact of rule sets on system performance within embedded environments, attention has been directed towards devices such as IoT or wireless devices when formulating search terms. The suggested search terms are provided below in no particular order:

1. Intrusion detection
2. Host intrusion detection system
3. Intrusion detection & Evaluation
4. Intrusion detection & Rule
5. Intrusion detection & IoT Devices

Databases

1. Web of Science
2. IEEE Xplore

Type of publications

1. Journal Articles
2. Proceeding papers

Academic journals provide in-depth, peer-reviewed insights that offer a solid foundation for understanding a subject. Conversely, conference proceedings enable the rapid dissemination of research findings, offering diverse viewpoints and fostering collaboration. While journals establish foundational knowledge, conference papers offer valuable insights into recent developments and emerging trends. However, it's worth noting, as emphasized by Levy and Ellis (2006), that conference papers may vary in quality. Therefore, scrutiny is essential to determine their suitability and reliability for the study.

Date of publications

The search process involved reviewing search terms from 2000 to 2023 to ensure comprehensive coverage of intrusion detection aspects, including older yet relevant findings.

2.5.4 Search for Literature

The process of searching and screening for relevant literature on intrusion detection involves using specific search terms, including "Intrusion detection," "Host intrusion detection system," "Intrusion detection AND Evaluation," "Intrusion detection AND rule," and "Intrusion detection AND IoT Devices." This search is carried out according to predefined inclusion and exclusion criteria, as outlined in the literature review table 2.1.

Inclusion criteria require that identified papers are authored in English, have undergone peer review (or are found in peer-reviewed publications through backward referencing), are published in recognized journals or conference proceedings, are freely accessible, were published between 2000 and 2023, and are available on IEEE Xplore

Inclusion Criteria	Exclusion Criteria
The paper has undergone peer review or is found in a peer-reviewed publication through backward referencing.	The papers without peer review or not found in peer-reviewed publications.
The paper is published in a recognized journal or conference proceeding.	The paper is not published in a recognized journal or conference proceeding.
The paper is freely accessible to the researcher.	Access to the paper requires payment.
The paper's abstract aligns with the research topic.	The paper's abstract is irrelevant to the research topic or does not address any of the stated research questions.
The paper is available on IEEE Xplore or Web of Science.	The paper is not available on IEEE Xplore or Web of Science.
The paper was published between 2000 and 2023.	
The paper is authored in English.	Redundant articles.

Table 2.1: Inclusion and exclusion criteria for the literature review

or Web of Science. Additionally, the abstracts of these papers must align with the research topic.

The exclusion criteria include disregarding redundant papers, as well as those lacking peer review or not found in peer-reviewed publications. Additionally, papers not published in recognized journals or conference proceedings, papers requiring payment for access, and papers not available on IEEE Xplore or Web of Science are excluded. Abstracts that are irrelevant to the research topic or do not address the stated research questions are also excluded.

By employing the specified search terms and applying the inclusion and exclusion criteria to assess research papers, the number of results corresponds to the selected articles based on their titles, representing those considered potentially relevant. The count of saved articles denotes those ultimately chosen for abstract review following the removal of duplicates and careful consideration of their content for our thesis. The literature search began with querying the Web of Science database using designated search terms, yielding results as outlined in table 2.2.

Database	Search Term	No. of results	No. of articles saved
Web of Science	"Intrusion detection"	28	4
Web of Science	"Host intrusion detection system"	10	2
Web of Science	"Intrusion detection" and "Evaluation"	9	1
Web of Science	"Intrusion detection" and "rule"	16	4
Web of Science	"Intrusion detection" and "IoT Devices"	5	1

Table 2.2: Results from the Web of Science literature search

The identical process was replicated using the second database, IEEE Xplore. Table 2.3 displays the findings from the literature search conducted on IEEE Xplore.

Database	Search Term	No. of results	No. of articles saved
IEEE Xplore	"Intrusion detection"	25	10
IEEE Xplore	"Host intrusion detection system"	32	8
IEEE Xplore	"Intrusion detection" and "Evaluation"	9	0
IEEE Xplore	"Intrusion detection" and "rule"	21	14
IEEE Xplore	"Intrusion detection" and "IoT Devices"	24	9

Table 2.3: Results from the IEEE Xplore literature search

Following the completion of the literature search, a total of 53 articles were saved for further analysis.

2.5.5 Extract Data and Quality Assessment

The literature search resulted in the identification of 53 documents, which were subsequently downloaded and reviewed. After careful examination, 41 documents were excluded based on specific exclusion criteria. These criteria included eliminating redundant papers, particularly those covering intrusion detection system techniques utilizing machine learning for rule generation, and those lacking peer review or not found in peer-reviewed publications. Additionally, papers not published in recognized journals or conference proceedings were excluded. Abstracts that were irrelevant to the research topic or failed to address the stated research questions were also excluded. To ensure the inclusion of only high-quality, data-driven studies focusing on aspects

such as system performance and ruleset impact, a thorough quality assessment was conducted. The final selection comprised 13 papers, listed as SLR results below in the table 2.4.

While assessing the articles, encountered a relevant study (Sen 2006) that may not have strictly adhered to academic or peer-reviewed standards. Nevertheless, it was included due to its data-driven analysis of the relationship between varying bandwidth and ruleset size.

The quality assessment emphasized studies with quantitative analysis, which collected data on rulesets and conducted analyses to identify trends, particularly focusing on managing rulesets in resource-constrained devices. These studies laid the groundwork for deeper exploration in the research. Selected papers were distinguished by their data-driven approach and conclusions, forming the basis for the ongoing investigation.

No	Reference
1	Arshad, J., Azad, M. A., Abdeltaif, M. M., & Salah, K. (2019). An intrusion detection framework for energy constrained IoT devices. Mechanical Systems and Signal Processing, 106436. https://doi.org/10.1016/j.ymssp.2019.106436
2	Asad, H., & Gashi, I. (2018). Diversity in Open Source Intrusion Detection Systems. Lecture Notes in Computer Science, 267–281. https://doi.org/10.1007/978-3-319-99130-6_18
3	Bures, M., Klima, M., Rechtberger, V., Ahmed, B. S., Hindy, H., & Bellekens, X. (2021). Review of Specific Features and Challenges in the Current Internet of Things Systems Impacting their Security and Reliability. ArXiv (Cornell University). https://doi.org/10.48550/arxiv.2101.02631
4	Ji, Y., Li, Q., He, Y., & Guo, D. (2015). Overhead Analysis and Evaluation of Approaches to Host-Based Bot Detection. International Journal of Distributed Sensor Networks, 11(5), 524627–524627. https://doi.org/10.1155/2015/524627
5	Khaled Sayed Soliman, Sobh, M. A., & Bahaa-Eldin, A. M. (2021). Survey of Machine Learning HIDS Techniques. https://doi.org/10.1109/icces54031.2021.9686138
6	Liu, Q., Veit Hagenmeyer, & Keller, H. B. (2021). A Review of Rule Learning-Based Intrusion Detection Systems and Their Prospects in Smart Grids. 9, 57542–57564. https://doi.org/10.1109/access.2021.3071263
7	Thongkanchorn, K., Ngamsuriyaroj, S., & Visoottiviseth, V. (2013, October 1). Evaluation studies of three intrusion detection systems under various attacks and rule sets. IEEE Xplore. https://doi.org/10.1109/TENCON.2013.6718975
8	Turner, C., Jeremiah, R., Richards, D., & Joseph, A. (2016). A Rule Status Monitoring Algorithm for Rule-Based Intrusion Detection and Prevention Systems. Procedia Computer Science, 95, 361–368. https://doi.org/10.1016/j.procs.2016.09.346
9	Vermeer, M., Michel van Eeten, & Gañán, C. (2022). Ruling the Rules: Quantifying the Evolution of Rulesets, Alerts and Incidents in Network Intrusion Detection. Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security. https://doi.org/10.1145/3488932.3517412
10	White, J. A., Fitzsimmons, T., & Matthews, J. (2013). Quantitative analysis of intrusion detection systems: Snort and Suricata. Proceedings of SPIE. https://doi.org/10.1117/12.2015616
11	Zarpelão, B. B., Miani, R. S., Kawakani, C. T., & de Alvarenga, S. C. (2017). A survey of intrusion detection in Internet of Things. Journal of Network and Computer Applications, 84, 25–37. https://doi.org/10.1016/j.jnca.2017.02.009
12	Sen, S., Koo, J., & Bagchi, S. (2018). TRIFECTA: Security, Energy Efficiency, and Communication Capacity Comparison for Wireless IoT Devices. IEEE Internet Computing, 22(1), 74–81. https://doi.org/10.1109/mic.2018.011581520
13	Sen, S. (2006). Performance Characterization & Improvement of Snort as an IDS.https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.720.2007

Table 2.4: Web of Science findings (1-11) and IEEE Xplore finding (12)

2.5.6 Synthesis Studies

The synthesis of several research studies reveals pivotal insights across three core themes: intrusion detection systems and security challenges in the Internet of Things, performance evaluation and the impact of rulesets on these systems, and machine learning techniques for advanced threat detection within the realm of intrusion detection and cybersecurity.

Intrusion Detection Systems and Security Challenges in the Internet of Things

Zarpelão et al. (2017) conducted a survey to present an overview of IDS research specific to IoT. The objective was to identify key trends, unresolved issues, and future research opportunities. The study analyzed 18 papers published between 2009 and 2016, proposing a taxonomy to classify IDS schemes for IoT based on detection methods, IDS placement strategies, specific threats, and validation strategies. This taxonomy and analysis contribute significantly to understanding IDS in IoT contexts, highlighting the importance of developing robust IDS frameworks tailored to IoT environments.

Similarly, Bures et al. (2021) conducted a comprehensive study examining the features and challenges impacting the security and reliability of IoT systems. This research combined a literature review with practical experiments on various IoT projects to identify key issues. The study highlighted twenty specific features, categorized into five main areas: economic, management, and organizational aspects; infrastructural challenges; security and privacy issues; complexity challenges; and interoperability problems. This detailed analysis provided valuable insights into the root causes of these challenges, their interconnections, and their potential effects on the overall quality of IoT systems.

Building on this, Sen, Koo, and Bagchi (2018) compared various wireless IoT devices in terms of security, energy efficiency, and communication capacity. The technologies surveyed include Wi-Fi, Bluetooth, Zigbee, NFC, and wide-area wireless technologies. The findings indicated that current battery technology is sufficient for low-performance devices but inadequate for high-performance ones, underscoring the need for improved communication energy efficiency. The study also emphasized the importance of enhanced security for smart devices, suggesting measures such as NFC-based secondary authentication, isolating hardware and software modules, and employing federated identity management to improve confidentiality, integrity, and availability.

These studies collectively offer a thorough examination of the current state and future directions of IoT security and intrusion detection. They underscore the importance of developing tailored IDS frameworks, enhancing energy efficiency, and addressing complex infrastructural and interoperability challenges to ensure the security and reliability of IoT systems.

Performance Evaluation and Impact of Rulesets on Intrusion Detection Systems

The approach presented by Arshad et al. (2019) introduces a novel framework for intrusion detection, combining host-based and network-based methods to achieve

efficient intrusion detection for IoT environments. This approach distributes intrusion detection tasks between IoT devices and an edge router, with IoT devices acting as IDS nodes. They scan packets to determine if they are malicious or legitimate, periodically sending alerts to the edge router about any malicious activity. The system's performance was evaluated through simulations of different network scenarios using the Contiki operating system. The results demonstrate that the proposed approach minimizes overhead in terms of energy consumption and memory usage, proving effective within IoT device constraints.

In a related study led by Ji et al. (2015), the focus was on identifying the factors influencing overhead in the development of host-based bot detection systems. The researchers categorized common approaches to host-based bot detection based on various metrics, considering the nature of bots—software programs designed to automate tasks, often with malicious intent. These approaches encompassed aspects such as information sources, interception mechanisms on the host, intercepted system calls, trigger mechanisms, and correlation engines. To investigate these factors, the researchers conducted experiments within a controlled environment. They set up four independent hosts, among which three were intentionally infected, while one acted as the botmaster. All hosts were interconnected within the same network. Through these experiments, the study unveiled three primary factors significantly impacting the overhead of host-based bot detection: the interception mechanism on the host, the type and volume of intercepted system calls, and the efficiency of the correlation engine.

Drawing insights from their findings, the researchers proposed strategies to mitigate the overhead associated with host-based bot detection. These strategies included streamlining the time and space complexity of correlation methods and optimizing calculation modules. Such optimizations are crucial for enhancing the efficiency and effectiveness of bot detection systems, thereby bolstering cybersecurity measures in network environments.

Furthermore, White, Fitzsimmons, and Matthews (2013) conducted a quantitative analysis to compare the performance of the intrusion detection systems Snort and Suricata. They created a testing framework to assess both systems, specifically examining how performance scales with varying system resources. The findings revealed that Suricata significantly outperformed Snort, even on a single core. Additionally, the study highlighted that variations in rulesets could lead to notable differences in performance. Suricata also demonstrated lower average memory usage and CPU utilization compared to Snort.

Moreover, Thongkanchorn, Ngamsuriyaroj, and Visoottiviseth (2013) evaluates three intrusion detection systems (IDS) under various attack scenarios and rulesets. Utilizing standard rulesets from the open-source community project Emerging Threats, which offers rulesets for IDS tools Snort, Suricata, and Bro, the study assesses the performance of these IDS under normal and malicious traffic generated by eight types of attacks. The results reveal that the number of alerts generated by an IDS is significantly influenced by the choice of ruleset.

In a similar vein, Sen (2006) explores the relationship between bandwidth and ruleset size in the performance of the Snort IDS. The experiment involved using a hardware-based packet generation tool to simulate different data rates by injecting

packets into the system. The results revealed a decrease in the number of rules supported as the incoming data rate (bandwidth) increased. The analysis indicated that larger rulesets imposed more severe time constraints on packet handling and pattern matching by Snort. This growing trend could lead to severe performance deterioration and packet loss if not adequately addressed.

Asad and Gashi (2018) analyzes the diversity of open-source intrusion detection systems, focusing on Snort and Suricata. Over five months, they evaluated the rulesets and blacklisted IP addresses of these two IDS. The study focused on four different default rule configurations: community rules, registered rules, subscribed rules, and the ET Open Suricata ruleset. They examined the diversity and overlap between ET's and Snort's IP blacklists and rule content options. The findings revealed that only 1% of the rulesets contained matching content options, resulting in minimal overlap in the alert-triggering behavior of the rulesets from both sources.

Furthermore, in a study conducted by Turner et al. (2016), the focus was on the management issues of rule-based intrusion detection systems (IDS), specifically addressing the problem of determining the enabled and disabled status of rules. An IDS can detect a particular event only if it has a corresponding rule that is enabled. To monitor the enabled and disabled status of 118 rulesets stored in the rules folder, the research developed an algorithm that searches through each ruleset (sets of rules with similar characteristics) to determine whether each rule is enabled or disabled and reports the total number of enabled and disabled rules. This algorithm was tested on the Snort intrusion detection system. The statistical analysis revealed that most rules are inactive by default, and it was found that the number of rules is growing significantly, raising concerns about rule management and system efficiency.

Lastly, in a study by Vermeer, Eeten, and Gañán (2022), they collaborated with a managed security service provider to gain insights into the evolution of IDS rulesets, the alerts they trigger, and the incidents investigated. They analyzed a comprehensive ruleset that included both commercial and proprietary rules, totaling 130,000 rules. The research involved accessing alert logs received by the Security Operations Center (SOC) from mid-2009 to mid-2018. The results showed that the vast majority of rules failed to produce a single alert, with 80% of all alerts being triggered by only 0.5% of the rules. The study also noted that rule developers frequently add new rules but rarely modify existing ones. This finding underscores the importance of optimizing ruleset management for more effective intrusion detection.

The studies collectively highlight the critical role of ruleset configuration in IDS performance evaluation. The choice of ruleset directly influences the detection capabilities, accuracy, and efficiency of IDS systems. Therefore, selecting appropriate rulesets tailored to specific environments and threat landscapes is essential for maximizing IDS effectiveness and enhancing cybersecurity defenses.

Machine Learning Techniques for Advanced Threat Detection

In a study focused on detecting unknown attacks in critical infrastructure like smart grids, researchers Liu, Hagenmeyer, and Keller (2021) investigated intrusion detection rules using rule learning techniques. These rules are periodically updated to adapt to the dynamic behavior of smart grid systems. Additionally, the study featured

a comprehensive survey of rule learning-based intrusion detection systems (IDS), offering in-depth insights into this research area. Moreover, it thoroughly analysed the suitability of various rule-learning techniques for IDS in smart grids. Furthermore, the study identified key criteria for developing effective intrusion detection rules and assessed their quality.

In a similar effort to detect zero-day threats, researchers Soliman, Sobh, and Bahaa-Eldin (2021) discussed and compared recent techniques that use machine learning at the host level to identify advanced cybersecurity attacks based on their behavioral impact on computers. These techniques employed various supervised and unsupervised learning methods, highlighting the importance of these approaches in enhancing the performance of host-based intrusion detection systems.

These studies collectively contribute to advancing intrusion detection capabilities by addressing the challenges posed by unknown and zero-day threats in critical infrastructure and host-based systems. By leveraging rule learning techniques and machine learning methods, these studies provide valuable tools and methodologies for enhancing the security posture of critical systems.

3. Method

This chapter begins with the experimental setup and system requirements essential for demonstrating the practical implementation. Subsequently, it delves into the process of threat modeling and describes the research methodology utilized in this study.

3.1. Experimental Setup and System Requirements

In this research, an experimental setup was utilized to demonstrate the practical implementation of the proposed method. The experimental environment included a local Microsoft Windows 11 Home system with 16GB of installed RAM and the Hyper-V virtualization extension. Intrusion and detection tasks were executed within a controlled environment using Oracle VM VirtualBox 6.1.

Two virtual machines were installed in the Oracle VM VirtualBox: one for performing intrusion activities as the attacker machine and the other for detecting intrusions as the target machine, both connected to the same VM network shown in the below figure 3.1. This experimental setup aimed to showcase the effectiveness of the implementation in real-world scenarios, where the target machine detects the intrusion attempts from the attacker machine and dynamically expands its rulesets based on the alerts triggered in the system. The use of virtualization technology enabled the simulation and analysis of various intrusion scenarios while maintaining the security and integrity of the experimental environment.

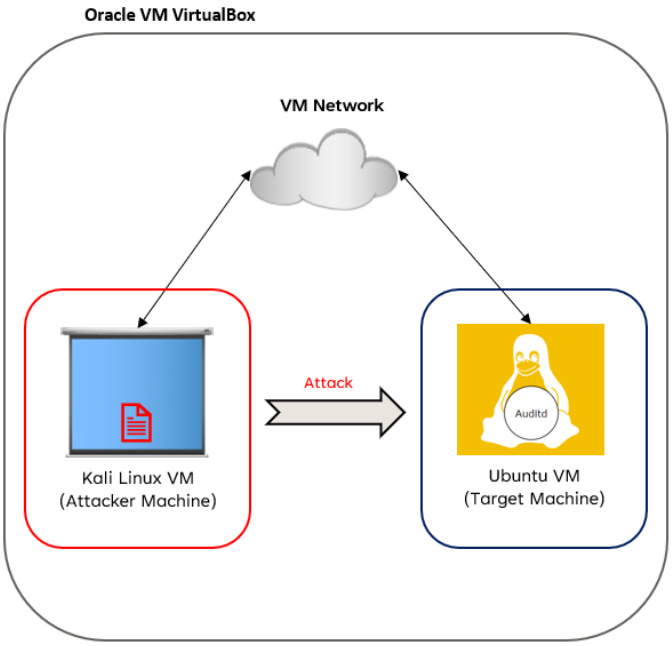

Figure 3.1: Experimental Setup

3.1.1 System Requirements

To experiment successfully, it is essential to ensure that both the local machine, running Microsoft Windows 11 Home, and the virtual machines (VMs) meet specific system requirements. Below are the detailed system requirements necessary for implementing the experiment:

Kali Linux VM

- Operating System: Kali Linux
- Kernel: x86_64 Linux 6.4.0-kali3-amd64
- CPU: 11th Gen Intel Core i7-11800H @ 4x 2.304GHz
- RAM: 10913 megabytes(MB)
- STORAGE: 67.1 gigabytes(GB)

Ubuntu VM

- Operating System: Ubuntu 16.04 xenial
- Kernel: x86_64 Linux 4.15.0-142-generic
- CPU: 11th Gen Intel Core i7-11800H @ 2.304GHz
- RAM: 9861 megabytes(MB)
- STORAGE: 26.2 gigabytes(GB)

3.1.2 AuditD: A Host Intrusion Detection System (HIDS) Tool

To detect intrusion at the host level using HIDS, the Linux auditing system was employed on the target machine. This system enhances system security by providing detailed insights into system activities. However, it does not directly enhance security by itself; it does not protect the system from code malfunctions or exploits. Instead, the Linux audit serves as a valuable tool for monitoring and tracking such issues, enabling users to implement additional security measures to mitigate risks (ArchWiki 2023). This includes monitoring system calls, file access, and directory modifications. The auditing mechanism comprises two primary components: the kernel audit subsystem and the user-space audit daemon (AuditD). The kernel subsystem generates audit records in response to system activity, which is then captured and logged by the userspace audit daemon (linux-audit/audit-kernel 2024). To enable AuditD to capture and log events, a predefined ruleset must be established beforehand. Additionally, these rules include filtering criteria to provide more context around specific system calls, such as restricting the analysis to calls made by non-root users (ArchWiki 2023).

Using AuditD as a HIDS in the experiment, based on the findings of Hashem and Zildzic (2022) performance evaluation of endpoint detection, proves suitable for HIDS deployment on the target machine. AuditD demonstrates efficient performance, particularly at low alert rates, and integrates a File Integrity Monitoring (FIM) module. It remains effective even with varying alert rates and exhibits scalability and resource

management capabilities. Moreover, the storage required to install AuditD on the system is less compared to intrusion detection tools like Wazuh and Falco (Hashem and Zildzic 2022).

3.2. Threat Modeling - Target Machine

Threat modeling entails a structured approach to recognizing and prioritizing potential threats to a system (OWASP Foundation —Threat Modeling 2022). It involves evaluating the effectiveness of different measures in mitigating threats. By conducting threat modeling during the project's design phase, significant resources can be conserved by pre-emptively addressing risks and integrating mitigations at an early stage. In safeguarding the target machine, threat modeling aids in analyzing the system from an adversarial viewpoint, concentrating on how attackers may exploit vulnerabilities within the system.

To aid in the design of threat modeling, the pytm framework (OWASP Foundation —Pythonic Threat Modeling 2024), developed by Tarandach (2024), was utilized. This open-source tool automates threat identification and assists in creating system models, and data flow diagrams (DFDs), through automated Python scripts. Unlike traditional methods, pytm primarily relies on code for creating and updating threat models. The source code available in appendix A.1, defines the model including its findings, and threats, with diagrams and reports considered outputs rather than inputs (OWASP Foundation —Pythonic Threat Modeling 2024). Moreover, it provides customization options and the flexibility to extend the framework to meet specific experiment setup requirements.

The target machine contains simulated confidential files accessible through SSH remote login. Additionally, there was a browser installed on the target machine, allowing access to the system's important files. The data flow diagram is depicted in figure 3.2.

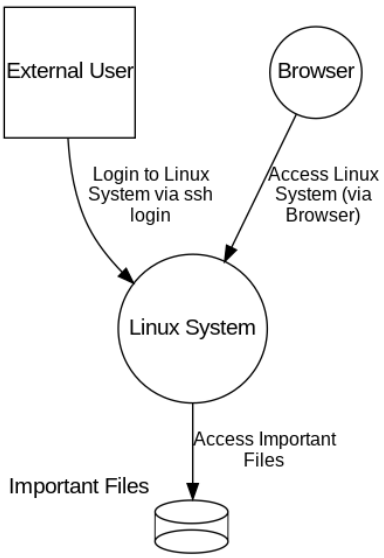

Figure 3.2: Data Flow Diagram

Based on the data flow diagram, the system's event sequence unfolds as follows: an external user accesses the Linux system through remote SSH logins. Additionally, an attacker gains access to the Linux system by exploiting vulnerabilities in the browser. Within the Linux system, simulated confidential files were present.

The pytm framework offers a comprehensive list of potential threats, with a focus on those relevant and present in simulated environments for successful intrusion and detection purposes within the designed threat model, as presented in Figure 3.3.

Threat Name	STRIDE Category	Priority
Server Side Include (SSI) Injection	Information Disclosure	Low
Command Line Execution through SQL Injection	Elevation of Privilege	Low
Authentication Abuse/ByPass	Elevation of Privilege	High
Excavation	Information Disclosure	Low
Double Encoding	Tampering	Low
Privilege Abuse	Elevation of Privilege	High
Flooding	Denial of Service	Medium
Path Traversal	Tampering	Low
Excessive Allocation	Denial of Service	Medium
Format String Injection	Elevation of Privilege	Low
Dictionary-based Password Attack	Information Disclosure	Medium
Audit Log Manipulation	Tampering	Medium
Data Breach(Important Files)	Information Disclosure	High

Figure 3.3: List of Threats

Additionally, these threats were categorized based on the STRIDE framework. The STRIDE framework, developed by Microsoft (Wikipedia 2024), plays a crucial role in this process. It derives its name from the mnemonic of each of its threat categories, namely Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege, as defined by Microsoft (Microsoft 2022). The descriptions of these categories are shown in Table 3.1.

Category	Description
Spoofing	It entails unlawfully obtaining and utilizing another user's authentication credentials, like their username and password.
Tampering	It encompasses the malicious modification of data. This can include unauthorized modifications made to stored data, like in a database, and changes made to data as it moves between two computers over an open network, such as the Internet.
Repudiation	It relates to users denying their actions without evidence, like performing unauthorized operations in a system without traceability. Non-Repudiation addresses this by ensuring systems can counter such threats. For instance, a user signing for a received package provides evidence against any denial of receipt.
Information Disclosure	It concerns the unauthorized access to information by individuals, such as users reading files, they shouldn't have access to or intruders intercepting data transmitted between computers.
Denial of Service	Aim to disrupt or make services temporarily unavailable to legitimate users. This can include rendering a web server unusable or inaccessible. Protection against DoS threats is crucial to ensure system availability and reliability.
Elevation of Privilege	It occurs when an unauthorized user gains privileged access, enabling them to compromise or potentially destroy the entire system. This scenario poses a significant risk, as the attacker may infiltrate system defences and gain trusted status within the system itself.

Table 3.1: Description of Threat Categories - STRIDE Model

Following the designed threat model and focusing on the sensitive information it contained and how the attacker could access the system, the prioritization of threats was established as follows:

High Priority: Threats categorized as high priority signify significant risks to system security and integrity, demanding immediate attention and resource allocation. These threats, such as "Authentication Abuse/Bypass," "Privilege Abuse," and "Data Breach(Important Files)," had the potential to cause severe damage or compromise sensitive information.

Medium Priority: Threats classified as medium priority indicate potential security concerns that require attention but might not pose an immediate or severe threat. While these threats, including "Flooding," "Excessive Allocation," "Audit Log Manipulation," and "Dictionary-based Password Attack," could lead to disruption or compromise, their impact was typically less severe than high-priority threats.

Low Priority: Threats designated as low priority were considered less critical or urgent compared to high and medium-priority threats within the designed threat model. These threats, such as "Server Side Include (SSI) Injection," "Command Line Execution through SQL Injection," "Excavation," "Double Encoding," "Path Traversal," and "Format String Injection," may have minimal impact on system security or functionality and may occur relatively infrequently or result in less damage.

If successful, the attacker gains direct access to the Linux system through remote SSH login, enabling them to execute commands, alter access levels, manipulate data, and potentially escalate privileged access. The most impactful consequence of this unauthorized access is the ability to access confidential files stored in the system, resulting in information disclosure. This could lead to severe breaches of privacy and compromise sensitive data.

Initially, the intrusion detection system identifies the attacker's entry point by monitoring authentication attempts, whether they originate externally or internally, with a specific focus on the SSH service. The ruleset should be designed to trigger when unauthorized login attempts occur. Additionally, additional rules need to be configured to detect the attacker's activities once they gain access to the system following a successful login.

To address the identified threat in the threat model, it's crucial to implement strong password policies and utilize an intrusion detection system for continuous monitoring and prompt investigation of suspicious activities. Regularly reviewing and updating access permissions, especially for sensitive files and directories, helps prevent unauthorized access. Furthermore, robust logging and monitoring mechanisms within the intrusion detection system aid in the timely detection and response to suspicious activities and unauthorized access attempts, ensuring effective mitigation measures.

3.3. Research Method

3.3.1 MITRE ATT&CK Framework

The research methodology was developed using the MITRE ATT&CK Linux Enterprise Matrix, specifically tailored for the Linux-based target machine. This matrix is renowned for its extensive collection of adversary tactics and procedures (TTPs), providing a structured analysis of attack behaviors derived from real-world observations (MITRE 2024). Leveraging this rich repository ensures a comprehensive understanding of attack patterns, encompassing descriptive analysis of existing behaviors outlined in the MITRE ATT&CK framework. ATT&CK serves as a resource for adversarial tactics, techniques, and common knowledge, emphasizing how adversaries interact with systems throughout an operation, covering various phases of the attack lifecycle and the platforms they target (MITRE ATT&CK —General Information 2024).

Tactics in the context of ATT&CK signify the strategic objectives pursued by adversaries when employing a technique or sub-technique. These objectives outline the motivations behind their actions, such as gaining access to credentials or compromising a system (MITRE ATT&CK —Tactics 2015).

Table 3.2 below, shows a comprehensive list of tactics paired with their corresponding objectives.

MITRE Tactics and Objectives
Initial Access: The adversary seeks to gain access to your network.
Execution: The adversary aims to execute malicious code.
Persistence: The adversary's objective is to maintain access to the network over time.
Privilege escalation: The adversary's goal is to obtain higher-level permissions.
Defense Evasion: The adversary is trying to avoid being detected by security measures, such as antivirus software or intrusion detection systems.
Credential Access: The adversary aims to steal account names and passwords.
Discovery: The adversary aims to gather information about your environment.
Lateral Movement: The adversary aims to move laterally through the environment.
Collection: The adversary is attempting to collect data relevant to their objective.
Command and Control: The adversary seeks to establish communication with compromised systems for control purposes.
Exfiltration: The adversary aims to exfiltrate data from your network.
Impact: The adversary aims to impact systems and data by manipulating, interrupting, or destroying them.

Table 3.2: MITRE Tactics and Objectives

MITRE techniques detail the specific strategies adversaries use to achieve their strategic objectives (MITRE ATT&CK —Techniques 2023). For example, one tactic involves credential dumping, where adversaries attempt to harvest authentication credentials from compromised systems (MITRE - Credential Dumping 2012).

Each tactic encompassed a variety of techniques. Since the designed threat model was centered around a Linux machine, utilized an open-source ruleset sourced from GitHub (bfuzzy 2024), specifically a Linux AuditD ruleset mapped to MITRE's ATT&CK Framework. Within this open-source repository, rules were listed to detect specific techniques associated with each tactic, aimed at identifying intrusion techniques used by attackers. This open-source approach facilitated the testing of a comprehensive ruleset tailored for Linux-based systems and simplified the creation and simulation of test cases for detecting threats on the target machine. Furthermore, leveraging this approach contributed to addressing the research question by testing a novel segmentation and dynamic expansion of IDS rulesets.

3.3.2 Segmentation and Dynamic Expansion Technique

The MITRE framework for Linux (MITRE ATT&CK Enterprise 2024) tactics was methodically divided into three distinct groups to facilitate a systematic approach to threat detection. The first group focused on tactics related to gaining initial access. This stage marks the beginning of an attack lifecycle, where adversaries attempt to breach the system's defenses and establish a foothold for subsequent activities. To detect these activities, an initial detection ruleset was employed. This ruleset encompasses both external and internal threats, including potential insider threats. For example,

within the designed threat model, it is crucial to detect unauthorized access via remote SSH logins, requiring the implementation of an initial detection ruleset triggered by events indicative of such activity.

Moving to the second group, MITRE tactics such as execution, persistence, and privilege escalation were grouped together. These tactics represent pivotal stages beyond the initial breach, requiring elevated access or system modifications to advance the attack. For example, post-infiltration, attackers may seek to elevate their privileges to manipulate system configurations. This highlights the importance of monitoring for attacker's behavior indicative of these tactics.

Intrusion detection initiates with the availability of data sources, which serve as the foundation for identifying actions and sequences of techniques executed by adversaries. According to the data-driven analysis conducted by Rajesh et al. (2022), the top 10 data sources defined for enterprises indicate that file monitoring ranks among the highest-utilized techniques. Therefore, a critical aspect of this defense strategy involves meticulously monitoring specific directories within the Linux system architecture, as shown in the below table 3.3 represent the prevalent top-level directories linked to the root directory (GeeksforGeeks 2021).

Directories	Description
/bin	Contains binary or executable programs. These programs are essential for basic system functionality and are often required during system startup.
/etc	Contains system configuration files. These files configure various aspects of the system, such as network settings, user permissions, and system services.
/home	Home directory. It is the default current directory for users. Each user has their own home directory, where they can store personal files and settings.
/opt	Contains optional or third-party software. This directory is typically used for installing software that is not provided by the operating system's package manager.
/tmp	Temporary space, typically cleared on reboot. This directory is used for temporary file storage by applications and the system. Files stored here are often disposable and are deleted automatically.
/usr	Contains user-related programs and data. This directory houses most of the system's user-installed software, including applications and libraries.
/var	Contains log files and other variable data files. This directory stores system logs, temporary files, and other variable data that may change during system operation.
/boot	Contains all the boot-related information files and folders such as configuration files and boot loader files. This directory is crucial for booting the system and managing the boot process.
/lib	Contains kernel modules and shared libraries needed for the system and its applications to operate smoothly. These libraries provide essential functions and services to applications and are shared among multiple programs.
/sbin	Contains binary executable programs specifically intended for use by system administrators. These programs perform administrative tasks and system maintenance functions, such as system configuration and network management.

Table 3.3: Description of Linux File Directories

By focusing on pivotal directories like /etc, /usr, and /var, along with critical system files, security configurations, system binaries, libraries, user and group activities, and system backups detailed in the table 3.4, which were act as the choke points to detect intrusions under the second group, along with execution, persistence, and privilege escalation detection rules. These choke points denote critical areas targeted by attackers seeking access or making modifications to achieve their objectives, which may vary depending on the attacker's goals.

Directory or File	Critical System Operation
/etc, /usr, and /var	High-level directories crucial for detecting unauthorized modifications and potential data exfiltration attempts. Specific subdirectories within /etc, such as /etc/passwd and /etc/shadow, are commonly targeted for credential theft or configuration manipulation.
User and Group Activity	Servers as choke points for monitoring authentication events, user home directories, and user account management. Attackers may target user home directories (/home) to steal login credentials or gain unauthorized access by modifying user account files (/etc/-passwd, /etc/group).
Critical System Files	Includes directories such as /boot, /sbin, and /lib, monitored for unauthorized modifications or access attempts. Tampering with these files can lead to unauthorized access, execution of malicious code, or injection of malware to achieve persistence.
Security Configuration	Files in /etc/security monitored for changes compromising system security posture, weakening security controls, or bypassing access restrictions.
System Binaries and Libraries	Stored in /bin, /lib, and /lib64, are monitored for unauthorized modifications leading to code execution, privilege escalation, or installation of malware.
System Backups	Directories like /var/backups are critical for data recovery but are also targeted by attackers to prevent system restoration or conduct ransomware attacks. Monitoring for changes or unauthorized access is essential to prevent data loss and financial damage from ransomware attacks.
sudo	The "sudo" command is essential for granting temporary administrative privileges to regular users, enabling them to execute specific commands with elevated permissions. It is crucial to monitor the usage of "sudo" to prevent unauthorized access or misuse of privileged commands. By default, in the system, the directories /etc, /bin, /boot, /lib, /lib64, /sbin, /usr, /var, and /srv require sudo privileges for access.

Table 3.4: Critical System Operations and Directories

For instance, in the case of malware, once it gains access to the system, it may exploit /etc configurations to embed itself within the /etc folder (Cozzi et al. 2018). These choke points are vital for intensive monitoring and surveillance efforts, enabling early detection of unauthorized modifications, potential data exfiltration attempts, and other malicious activities initiated by threat actors (MITRE — Data Source 2024). Monitoring these critical directories enhances threat coverage and bolsters system security by addressing critical operations that exist in the Linux system.

The study by Cozzi et al. (2018), analyzing 10,000 Linux malware samples, emphasizes the critical role of identifying choke points in the Linux system, such as file paths, for detecting persistence techniques. It underscores the importance of monitoring these critical directories and operations in relation to attacker objectives, especially malware. The findings presented in this large-scale analysis (Cozzi et al. 2018) revealed that malware commonly exploited /etc configurations to embed itself within the /etc folder and create symbolic links to directories with run-level configurations. Additionally, a time-based execution approach to persistence was designed to modify configuration files stored under the /var folder. Significantly, the study highlighted the top ten accesses on the configuration file /etc/ in Linux based on the analysis of the malicious samples. In Linux systems, cron configuration files are scheduled to execute at fixed time intervals, demonstrating a tailored approach to modifying configuration files under the /var folder and concealing their traces by deleting all log files in /var/log (Cozzi et al. 2018).

Moreover, malware utilizes file infection and replacement techniques, targeting binaries in /bin/ or /usr/bin/ folders for system-wide infection (Cozzi et al. 2018). Thus, identifying and monitoring critical file paths serves as crucial choke points for detecting and mitigating persistence techniques employed by malicious actors.

The third group encompasses a range of advanced tactics, including defense evasion, credential access, discovery, lateral movement, collection, command and control, exfiltration, and impact. These tactics, along with their corresponding techniques, represent sophisticated strategies employed by attackers to advance their objectives within compromised systems. The advanced detection ruleset is tailored to identify and counteract these malicious activities (Google Security Operations 2024). This ruleset is designed to address scenarios such as accessing system files, which requires monitoring subsequent operations like data modification, deletion, or exfiltration attempts to proactively mitigate threats. Referred to as the data breach segment, these rules specifically target attacker objectives, particularly those involving breaching confidential files. The segment rules validate the threat landscape for confidential files and scrutinize all related activities.

In essence, advanced detection rules encompass detection mechanisms for defense evasion, credential access, discovery, lateral movement, collection, command and control, exfiltration, and impact tactics, covering the entire threat landscape. This approach broadens coverage across various threat scenarios, such as malware-related incidents, DDoS attacks, or data breaches, ensuring comprehensive detection capabilities.

The research methodology systematically organizes tactics and techniques into distinct groups as shown in the image 3.4, focusing on a structured and comprehensive

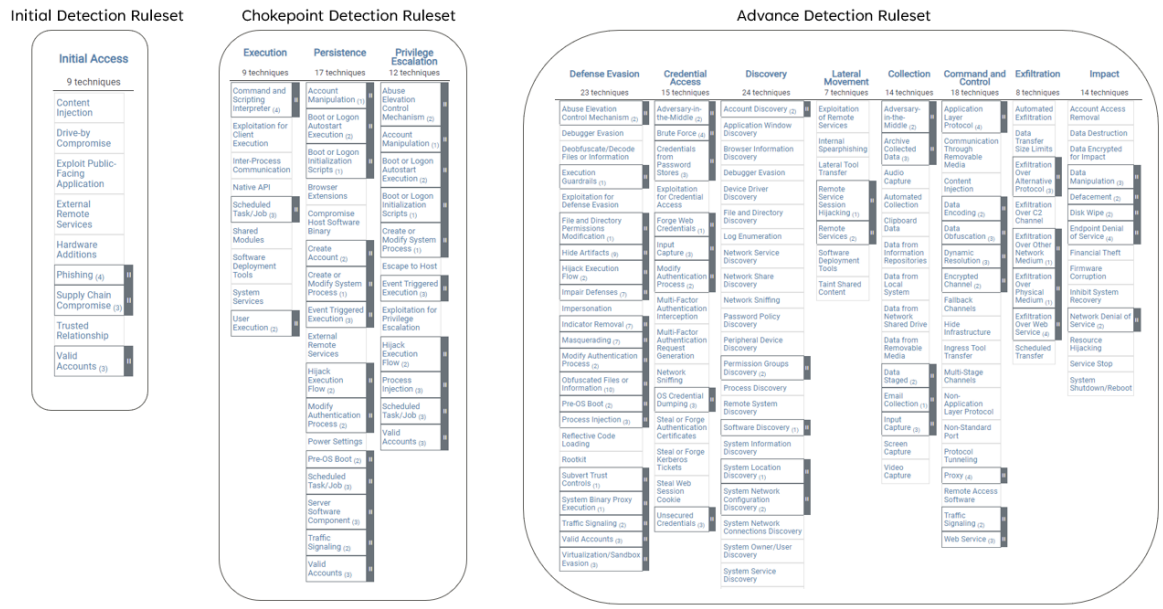

Figure 3.4: Grouping Tactics and their Techniques

approach to detecting both single and multi-stage attacks. The multi-stage attacks progress through a series of steps, including gaining an initial foothold, spreading across the internal network, and ultimately achieving specific objectives like data theft or operational disruption (Takey et al. 2021). Each stage of these multi-stage attacks is detected within their respective tactics and techniques, as outlined in the grouping method employed. By categorizing these attacks, the methodology aims to achieve thorough coverage of the threat landscape, effectively addressing potential vulnerabilities and attack vectors at each stage.

In the research methodology, a dynamic expansion approach as a rule addition based on the threat activities was utilized. This involved merging the initial detection ruleset and chokepoint ruleset to form what was referred to as the global ruleset. This consolidation enables a systematic approach to tracking multi-stage attacks, allowing us to detect each entry point of the attack—whether external or internal—and correlate subsequent activities as chokepoints aimed at achieving the attacker's goals or objectives.

When an alert was triggered by the combination of these initial detection ruleset and choke point ruleset, a relevant segment (a set of rules) was dynamically incorporated into the global ruleset in real time based on a logical formula. This method ensured comprehensive coverage of the threat landscape, enhancing detection capabilities in the event of an intrusion by adding suitable segments tailored to the respective attackers' objectives effectively.

3.3.3 Segment Selection Logic

To systematically categorize threat scenarios and enhance detection efficacy, a logical formula for segment selection was implemented. The process was designed to efficiently

assign threat scenarios to appropriate segments based on the alerts generated by the initial detection ruleset (I1, I2) and choke point detection ruleset (C1, C2, C3, C4, C5). The advanced detection ruleset (ADR1, ADR2, ADR3) were identified under Segment 1 (S1), while (ADR4, ADR5,ADR6) were identified under Segment 2 (S2).The logic unfolds below and the graphical representation in the figure 3.5.

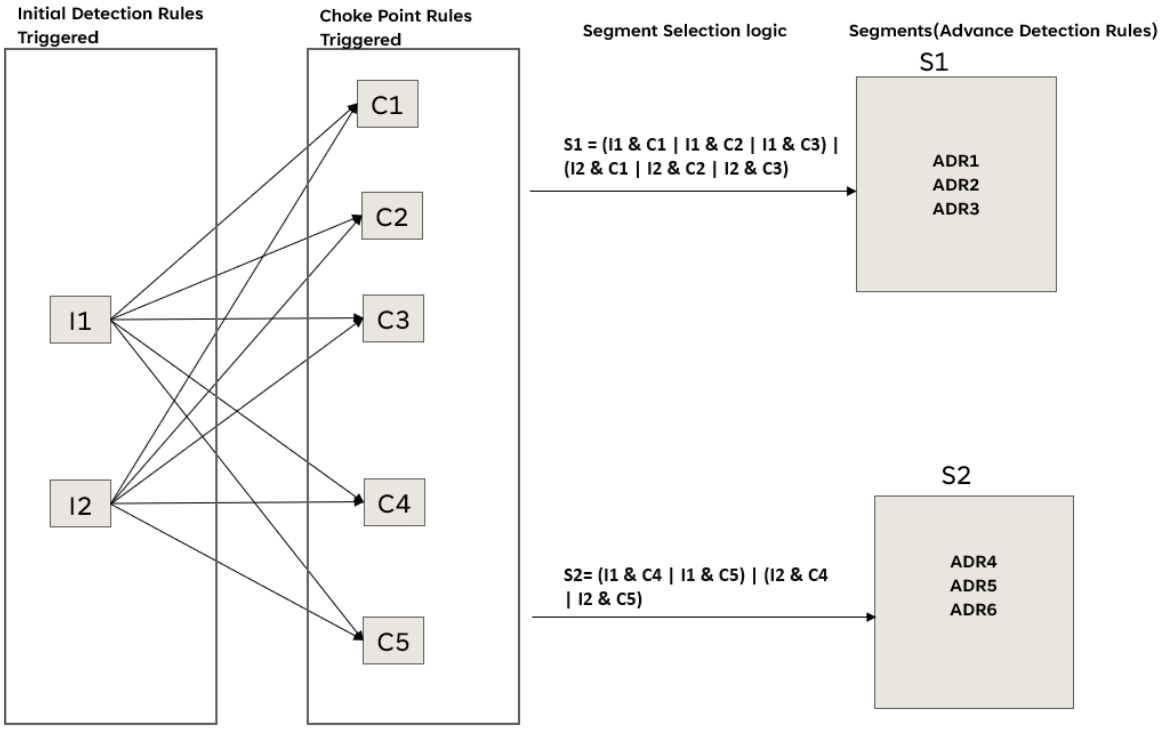

Figure 3.5: Segment Logic

Two distinct segments were defined to address specific threat objectives. Segment 1 focused specifically on data breach activity, aiming to identify and mitigate threats related to unauthorized access or manipulation of sensitive information. The choke points identified for Segment 1 were C1, C2, and C3, representing key areas where potential breaches might occur or where sensitive data could be accessed or exfiltrated. This segmentation approach enabled a more targeted and effective detection strategy to address distinct threat scenarios within the system.

Segment 2 was designed to detect activities related to malware, targeting potential intrusion indicators such as system modifications or suspicious processes. The choke points identified for Segment 2 included C4, and C5, which served as critical areas for monitoring and detection within this context.

Rule Triggering: The process begins with the activation of initial detection rules and choke point detection rules, reflecting potential threat activity within the system.

Segment Assignment Criteria

Segment 1 Selection: If either initial detection rule I1, for detecting external attack access, or I2, indicating the attacker's attempt to gain internal access, was triggered simultaneously with a choke point from the first set (C1, C2, C3), Segment 1 was assigned to the threat scenario.

If $(I1\&C1|I1\&C2|I1\&C3)|(I2\&C1|I2\&C2|I2\&C3)$ alerts were triggered,

Segment 2 Selection: Conversely, if any initial detection rule (I1 or I2) coincides with a choke point from the second set (C4, C5), Segment 2 is assigned.

Else if $(I1\&C4|I1\&C5)|(I2\&C4|I2\&C5)$ alerts were triggered,

The addition of the appropriate segment, including the advanced detection ruleset, enhances the global ruleset's capability to detect future attacker actions effectively. It's crucial to emphasize that each time an attacker's action occurs, it undergoes matching against all global ruleset. The logic guiding segment selection is both straightforward and efficient. For instance, when an attacker tries to remotely access the system, the initial ruleset identifies the login attempt, followed by subsequent actions triggering the choke point ruleset. This logic ensures that each initial rule aligns with the corresponding dedicated choke point to address the threat objective, minimizing the likelihood of errors. However, a drawback arises if the attacker's action fails to trigger an alert due to the absence of an initial or choke point rule defined in the global ruleset to detect it, resulting in a missed genuine threat.

The presented steps below illustrate the logic behind the approach adopted in the research methodology.

Step 1:

The global ruleset was created by manually adding the initial detection ruleset and choke point ruleset to the `/etc/audit/audit.rules` file. This file contains all the rules of the AuditD tool, which were responsible for matching intrusion patterns and detecting malicious activities.

Step 2:

The simple below bash script checks if both I1 and C1 exist as alerts, and selects the corresponding segment S1.

```
check_and_select_segment() {
    if [ alert_I1_exists ] && [ alert_C1_exists ]; then
        segment_selected="S1"
        echo "Segment $segment_selected selected based on alerts I1 and C1"
    elif [ alert_I2_exists ] && [ alert_C4_exists ]; then
        segment_selected="S2"
        echo "Segment $segment_selected selected based on alerts I2 and C4"
    fi
}
```

Step 3:

Add the advanced detection ruleset to the global ruleset.

```
add_advanced_ruleset() {
    if [ "$segment_selected" == "S1" ]; then
        echo "Adding rules for segment S1 to global ruleset"
        # Command to add rules for segment S1
    elif [ "$segment_selected" == "S2" ]; then
        # Add advanced detection ruleset S2 to global ruleset
    fi
}
```

3.3.4 Rationale for Methodology Selection

The methodology selected for this research was driven by the necessity to manage extensive rulesets effectively within HIDS, particularly considering the limitations of resource-constrained embedded systems. It also aimed to minimize performance overhead while ensuring robust security measures.

To address challenges related to embedded system resources, AuditD, a user-space component of the Linux Auditing System, was chosen due to its minimal resource requirements compared to alternative intrusion detection tools like Wazuh and Falco (Hashem and Zildzic 2022). The integration of AuditD with the kernel facilitates straightforward data collection, making it well-suited for HIDS activities.

The research methodology involves mapping the extensive ruleset to known adversary tactics, techniques, and procedures (TTPs) documented in repositories like MITRE (MITRE ATT&CK —Techniques 2023). This structured approach divides tactics into three groups: initial detection ruleset, choke point detection ruleset, and advanced detection ruleset, focusing on each group's tactics and outcomes to develop detection strategies for specific stages of the attack lifecycle and enabling more granular threat detection.

To streamline the management of the extensive ruleset, the methodology employs the initial detection ruleset and choke point detection ruleset, collectively referred to as the global ruleset. These rulesets play a crucial role in detecting an attacker's initial access to the system and subsequent actions, such as execution, persistence, and privilege escalation tactics, along with the corresponding techniques used by the attacker to achieve their objectives.

To comprehensively cover the threat landscape beyond detecting initial access and chokepoints, the methodology introduced Segments, which were sets of rules designed to detect specific attacker objectives during subsequent actions. A logical formula was devised to select the appropriate segment based on combinations of generated alerts. This approach ensured that only relevant segments were added to the ruleset, optimizing computational resources and enhancing real-time detection by matching against a smaller subset of rules.

The global ruleset continuously assesses the attacker's actions against its set of rules. When it detects a combination of the initial and chokepoint, it triggers dynamic

expansion by adding a segment below the global ruleset to address the threat landscape and the attacker's goals or objectives. This dynamic expansion capability allows the system to adapt to evolving threat landscapes by incorporating advanced detection rules based on attacker actions. Such flexibility ensures the system remains effective in detecting emerging threats and can promptly respond to new attack techniques.

The inclusion of test cases serves as a proof of concept, demonstrating the efficacy of the segmentation and dynamic expansion technique in simulation and verification scenarios. This empirical validation provides confidence in the effectiveness of the methodology and its potential real-world applicability.

MITRE ATT&CK transcends mere attack tactics; it serves as a comprehensive repository of adversarial tactics and techniques derived from real-world observations. It reflects the various phases of an adversary's attack lifecycle, including the platforms they target and potential defense strategies (MITRE 2024). However, the operational approach utilized for dynamic expansion is grounded in the sequence of attack tactics aimed at achieving success, thereby affirming the significance of the cyber kill chain (Naik et al. 2022).

3.3.5 Consideration of Alternatives

The chosen novel approach to managing the extensive ruleset entails maintaining a global ruleset that continuously set to match threat actions, supplemented by logical segmentation selection to comprehensively address the threat landscape. While Vermeer, Eeten, and Gañán (2022) explored data-driven research, offering insights into the analysis of large rulesets, their approach lacked practical implementation strategies for managing extensive rules. Despite their suggestions, such as tailoring rules to cover specific threats in the threat model and updating rules triggered by corresponding alerts, the difference lies in the practical implementation demonstrated by this approach in HIDS systems.

To ensure the effectiveness of the presented approach, it was compared to alternative methodologies for detecting multi-stage attacks. Vasilomanolakis et al. (2016) proposed a multi-stage attack detection method using honeypots for Industrial Control Systems (ICS). While proactive, this approach relies solely on packet data signatures within ICS networks, potentially missing other attack vectors. Bhatt, Yano, and Gustavsson (2014) introduced a framework leveraging the Intrusion Kill Chain and defense patterns for multi-stage attack detection. Despite its structured approach, it overlooks adversary techniques critical for comprehensive detection. Mathew et al. (2005) offers real-time multi-stage attack detection based on intrusion detection sensor alerts, providing immediate threat identification. However, its focus on network traffic analysis may overlook attacks involving host-based tactics. Aparicio-Navarro et al. (2018) proposed a novel approach for identifying the Point of Entry (PoE) to a target system, achieving a high detection rate. Yet, it lacks an effective correlation between network and host events, limiting detection capability.

While all the presented multi-stage detection approaches offer valuable insights, they do not address the impact of rulesets on embedded systems or provide clarity on ruleset management effectiveness. In contrast, the presented methodology involves

mapping rulesets to adversary tactics for comprehensive detection, maintaining a selective approach for efficient detection, and adapting segments as needed to cover the threat landscape thoroughly. This comprehensive approach ensures both effective threat detection and reduced resource utilization in embedded systems, although it is restricted to HIDS on the Linux system.

3.3.6 Data Collection

The data collection for this research involved multiple stages and approaches to gather comprehensive information pertinent to the study's objectives. Initially, a systematic literature review was conducted to identify existing research studies, methodologies, and insights related to intrusion detection systems (IDS), rules, and IoT devices. Academic databases such as IEEE Xplore and Web of Science were extensively searched using relevant keywords and criteria to ensure the inclusion of recent and relevant publications, including journal articles and conference papers. Emphasis was placed on data-driven publications from the period 2000 to 2023 to ensure comprehensive coverage of relevant literature and insights.

For the research, an open-source GitHub repository (bfuzzy 2024) was utilized. This repository contained a Linux AuditD ruleset mapped to MITRE's ATT&CK Framework, comprising rules aligned with various tactics and techniques to detect intrusions initiated by attackers. The open-source ruleset was selected for simulation test cases. Segmentation and dynamic expansion of the ruleset were conducted on the target machine, with controlled attacks simulated from an attacker machine. Leveraging the AuditD ruleset installed on the target machine, intrusion attempts were effectively detected

3.3.7 Data Analysis Procedure

This research involved developing a data analysis procedure specifically designed to meticulously analyze audit logs generated during simulated cyberattack scenarios on the target machine. Furthermore, a performance evaluation of system call execution on the target machine was conducted as part of this study.

Segmentation and Dynamic Expansion Technique

To demonstrate the segmentation and dynamic expansion technique, a simulated cyberattack scenario was meticulously designed within a threat model, employing the high-priority threats from the list of threats generated by the pytm framework. These test cases were structured into multiple steps, each mirroring a specific phase of the attack lifecycle and originating from the attacker's machine. The primary objective of this analysis was to systematically scrutinize the audit logs generated on the target machine during these simulated attacks initiated by the attacker machine while evaluating the segmentation and dynamic expansion technique on the target machine. The adversarial Bash scripts for test cases executed by the attacker's machine are included in the appendix A.2 & A.3.

For the experiment, the global ruleset consisted of the initial and chokepoint detection rulesets derived from the Linux AuditD ruleset (bfuzzy 2024). Detailed

information about the full detection ruleset is presented in Appendix A.4. Segment 1 (S1) was designed to detect data breach activities related to accessing sensitive files, while Segment 2 (S2) contained rules to detect malware activities.

Firstly the global ruleset was created by manually appending the initial detection ruleset and choke point ruleset to the `/etc/audit/audit.rules` file. This file encompasses all the rules of the AuditD tool, responsible for detecting intrusion patterns and malicious activities. In the test cases described, all actions and time delays were executed on the attacker's machine, while the AuditD ruleset for detection and the expected outcomes were examined on the target machine.

Test Case 1: Sensitive File Access

This test case focused on the sensitive files stored on the target system. The test case involved simulating a cyberattack scenario with three sequential steps: In the first step, the attacker gains unauthorized access to the Linux system by remotely connecting via SSH using compromised credentials. This action triggers an alert in the audit logs, indicating a successful SSH login. Subsequently, the attacker attempts to access the sensitive file stored in the Linux system. AuditD, the Linux auditing tool, detects and logs this unauthorized access attempt. Based on the combination of the initial detection rule and the choke point detection rule, the first segment (S1) expands. Finally, the attacker executes commands to retrieve system information, such as the current user and disk usage, triggering additional alerts in the audit logs.

Step 1: Infiltration- The initial access ruleset was designed to detect SSH logins.

- Action: The attacker gains access to the Linux system by connecting to a remote machine via SSH using compromised credentials.
- Expected Outcome: AuditD generates an alert indicating a successful SSH login.
- AuditD Ruleset for Detection:
  `-a always,exit -F arch=b64 -S sshd -F success=1 -k ssh_login_pass`
- Time Delay: 5 seconds

Step 2: Access to Sensitive File - The ruleset includes monitoring sensitive file access, modifications to password files, and any new file creations, which serve as chokepoints to select Segment 1.

- Action: The attacker attempts to access a sensitive file by opening it.
- Expected Outcome: AuditD generates an alert indicating an attempt to access the sensitive file.
- AuditD Ruleset for Detection:
  `-w /home/khaldrogo/Desktop/login_info.txt -p`
  `r -k Important_file_access`
- Time Delay: 10 seconds

Step 3: Segment Selection - The combination of the initial access ruleset with the chokepoint was set to select Segment 1 and add it to the global ruleset.

This segment was specifically dedicated to monitoring system discovery activities and detecting data exfiltration techniques. These rules covered potential threat activities that could occur after an attacker gained system access and took subsequent actions. The segment selection logic script is available in the Appendix A.5.

Step 4: User and Group Information & Encoding Sensitive Data

- Action: The attacker searches for user and group information for the current user and performs other actions to encode the sensitive data.
- Expected Outcome: AuditD generates an alert indicating the use of the 'id' and 'base64' commands.
- AuditD Ruleset for Detection:
  `-w /usr/bin/id -p x -k T1087_Account_Discovery_id`
- Time Delay: 5 seconds
- AuditD Ruleset for Detection:
  `-w /usr/bin/base64 -p x -k susp_activity`

Test Case 2: Malware Activities

Simulates a cyberattack scenario with sequential steps: Infiltration, Privilege Escalation, and System Information Discovery. In the first step, the attacker gains unauthorized access to the Linux system by connecting remotely via SSH using compromised credentials. This action triggers an alert in the audit logs, indicating a successful SSH login. Subsequently, the attacker attempts to escalate privileges within the system by accessing the sudo configuration file, utilizing the sudo command. AuditD, the Linux auditing tool, detects and logs this attempt to access the configuration file. Based on the combination of the ssh login and the access to the sudo configuration file, the first segment (S2) expands. Finally, the attacker executes commands to retrieve system information, such as the current user and disk usage, triggering additional alerts in the audit logs.

Step 1: Infiltration - The initial access ruleset was designed to detect SSH logins.

- Action: The attacker gains access to the Linux system by connecting to a remote machine via SSH using compromised credentials.
- Expected Outcome: AuditD generates an alert indicating a successful SSH login.
- AuditD Ruleset for Detection:
  `-a always,exit -F arch=b64 -S sshd -F success=1 -k ssh_login_pass`
- Time Delay: 5 seconds

Step 2: Privilege Escalation - The ruleset included monitoring the system-wide configuration file and sudo privilege permissions, which served as chokepoints based on the study conducted by Cozzi et al. (2018). These rules were dedicated to selecting Segment 2.

- Action: Once inside the system, the attacker attempts to access the sudo configuration file by switching to the root user (sudo /etc/sudoers) with a delay of 5

seconds.

- Expected Outcome: AuditD detects and logs the sudo command for privilege escalation file access.
- AuditD Ruleset for Detection:
  `-w /etc/sudoers -p rwa -k Sudo_configuration_file`
- Time Delay: 10 seconds

Step 3: Segment Selection -The combination of the initial access ruleset with the chokepoint was used to select Segment 2 and integrate it into the global ruleset. This segment was specifically dedicated to monitoring system discovery activities and detecting data exfiltration techniques related to malware. These rules addressed potential threat activities that could occur after an attacker gained system access and performed subsequent actions. The script for segment selection logic can be found in Appendix A.5.

Step 4: System Information Discovery

- Action: The attacker executes commands to retrieve system information, including the current user (whoami) and disk usage (du), with a delay of 5 seconds between commands.
- Expected Outcome: AuditD logs alerts for the whoami and du commands executed by the attacker.
- AuditD Ruleset for Detection:
  `-w /bin/whoami -p x -k T1087_Account_Discovery_whoami`
- Time Delay: 5 seconds
- AuditD Ruleset for Detection:
  `-w /bin/du -p x -k T1083_File_and_Directory_Discovery_du`

These test cases were examined on the target machine based on audit logs containing alerts generated by AuditD in response to commands executed by the attacker machine. Attention was focused on the time delay mentioned in each log entry. The analysis proceeded sequentially through each step of the attack scenario. For each step, the audit logs were reviewed, and the timestamp associated with each alert was noted. The specified time delay in the audit log for each command executed by the attacker was compared with the actual time elapsed between consecutive command executions.

Performance Evaluation of Target Machine

AuditD monitors system calls on the Linux system, tracking requests made by user-level processes to the kernel for various services or resources (Wikipedia 2020). In the data-driven study (Hashem and Zildzic 2022), syscall execution time was measured as a performance benchmark. This measurement reflects the responsiveness of the system or application in completing tasks and processing data within a specified timeframe.

To evaluate the system performance of the target machine, the open-source AuditD full ruleset detailed in Appendix A.4 was utilized. This ruleset combines the

initial detection ruleset and the choke point ruleset into a global ruleset. As part of the evaluation, syscall execution time served as a metric when no rules were triggered. The assessment involved comparing system responsiveness under three conditions: Without AuditD, with AuditD containing the full ruleset, and with AuditD using the global ruleset.

For the evaluation, the 'getppid' syscall was utilized, responsible for retrieving the process ID of the parent of the current process (Linux — getppid 2024). This was achieved by repeatedly calling getppid in a loop running 10000 times in the target machine, with each syscall's execution time measured. The script is available in the Appendix A.6. Records were stored for each call, documenting the time taken to retrieve the parent process's ID. It's important to note that during the experiment, no other applications were running on either the virtual machine or local machine to ensure consistent results.

Additionally, syscall monitoring leads to significant overhead, as noted by Hashem and Zildzic (2022). To evaluate system performance under triggered rules, the open system call, responsible for file opening operations, was utilized (Linux — open 2024). The scenarios were designed to incorporate both the full and global ruleset, facilitating the measurement of syscall execution time in the Linux system while accurately recording the duration of each operation. The rule was set to trigger in the global ruleset, and in another case, in the full ruleset, when the open syscall was made to access the sensitive file. The program was executed in a loop for 10,000 cycles, with each iteration initiating an open operation followed by a subsequent close action. This sequence of openat/close syscalls allowed for the precise measurement of the system's execution time script is presented in the Appendix A.7.

3.3.8 Ethical Considerations

Ethical aspects of the research method do not apply to this study, as it does not involve human subjects or living components.

4. Results

This chapter presents the results of the test cases, demonstrating simulated attack scenarios and their expected outcomes, and highlighting the segmentation and dynamic expansion technique. Additionally, it includes an analysis of system performance data.

4.1. Segmentation and Dynamic Expansion Technique

The global ruleset configured in AuditD continuously detects intrusion patterns and malicious activities.

Test Case 1: Sensitive File Access

Simulated a cyberattack scenario consisting of three sequential steps: Infiltration, Access to Sensitive File, and System Information Discovery. Throughout these steps, the attacker sought unauthorized access to the Linux system, subsequently attempting to access sensitive files and retrieve system information.

Step 1: Infiltration - The attacker's machine successfully accessed the remote login of the target machine and logged in. AuditD generated the log at 21:48:35 on May 9, 2024, as shown in the log entry in the figure below 4.1, with the ruleset triggered upon the attacker's remote SSH access.

Figure 4.1: SSH_Login

Step 2: Access to Sensitive File - Five seconds after accessing the Linux system via remote SSH, the attacker used the 'cat' command to open the sensitive file, triggering the rule set in AuditD to generate an alert. This event occurred at 21:48:40 on May 9, 2024, as depicted in Figure 4.2.

Figure 4.2: Access to Sensitive File

Step 3: Segment Selection - The global ruleset, as depicted in the figure below, and marked by a yellow-colored rectangular box 4.3. Based on the initial rule triggered by the SSH login and access to sensitive files, the logic was set to add Segment 1. The selection and addition of the segment were facilitated by the script ./ruleadd.sh, which appends the S1 segment to the global ruleset. This process is illustrated by the green rectangular box 4.3 in the figure.

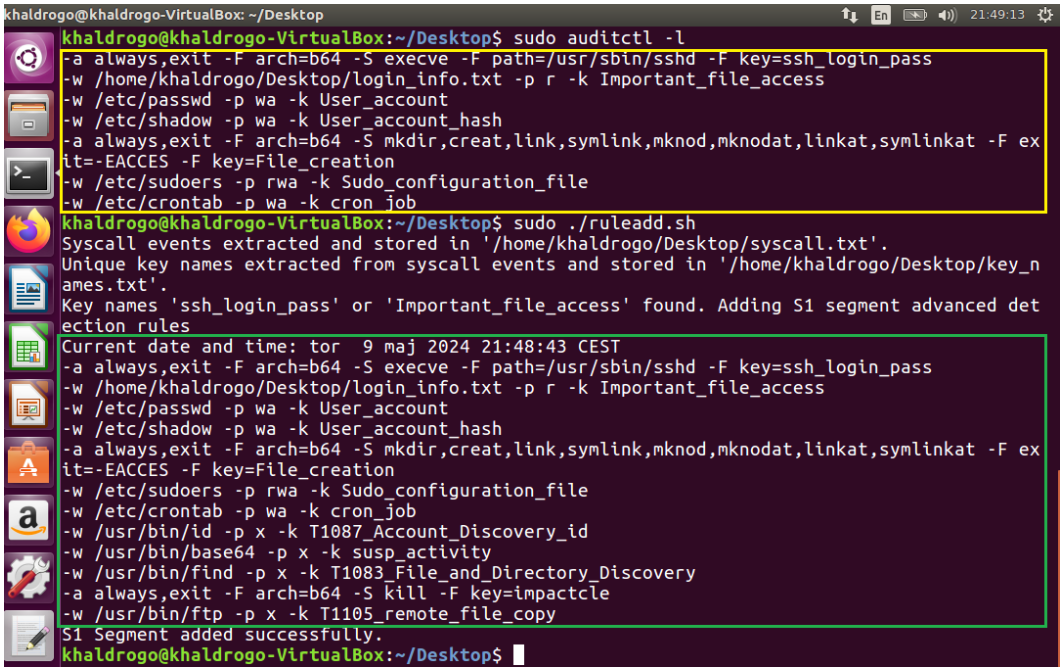

Figure 4.3: S1 Segment Addition

The segment S1 was added to the global ruleset at 21:48:43 on May 9, 2024.

Step 4: User and Group Information & Encoding Sensitive Data - After a 10-second delay, the attacker attempted to retrieve user and group information by executing the system administrator command 'id'. This action was part of the attacker's attempt to gather additional system details. AuditD triggered the alert based on the ruleset at 21:48:50 on May 9, 2024, as showed in the below figure 4.4.

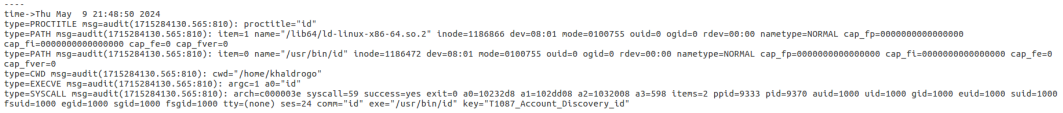

Figure 4.4: User and Group Information

Subsequently, the attacker waited 5 seconds before executing the 'base64' command to encode the sensitive file at 21:48:55, as showed in the figure below 4.5.

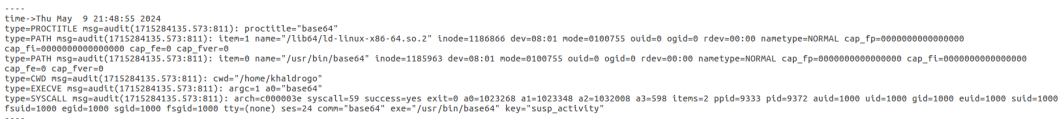

Figure 4.5: Encoding Data in Linux

The logic used to access the sensitive file triggered the addition of the S1 segment to the global ruleset. This segment identified system discovery information activities on the system. The dynamic addition of the S1 segment was based on the triggering of initial and choke point rules. Additionally, the timestamps of the attack delay were reflected in the AuditD logs, confirming the detection of a successful multi-stage attack on the target machine and providing evidence for subsequent actions within the system.

Test Case 2: Malware Activities

A cyberattack scenario was simulated, consisting of three sequential steps: Infiltration, Privilege Escalation, and System Information Discovery. Throughout these steps, the attacker attempted to gain access to the Linux system via SSH and subsequently sought to escalate privileges, followed by an attempt to retrieve system information for future attacks.

Step 1: Infiltration - The attacker's machine successfully accessed the remote login of the target machine and logged in. AuditD generated the log entry at 22:26:15 on May 9, 2024, as depicted in the image 4.6 below.

Figure 4.6: SSH_Login

Step 2: Privilege Escalation - After a 5-second delay following the initial phase of the attack, during which the attacker attempted to gain elevated privileges by executing 'sudo' to open the /etc/sudoers system-wide configuration file, the ruleset in AuditD was triggered, generating the log entry at 22:26:20 on May 9, 2024, as depicted in the image 4.7 below.

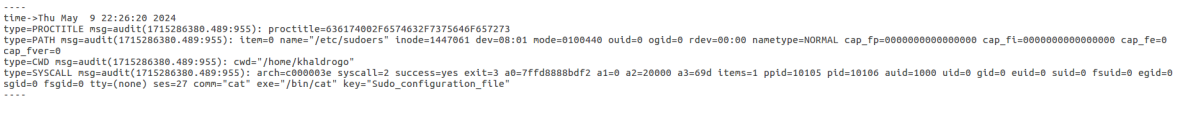

Figure 4.7: Privilege Escalation

Step 3: Segment Selection - The global ruleset was illustrated in the figure below and denoted by a yellow-colored rectangular box in the below image 4.8. Based on the initial and choke point rules triggered by the SSH login and sudo privilege escalation, Segment 2 was chosen for addition to the global ruleset. The selection and addition of the segment are determined by the script ./ruleadd.sh, which incorporates the S2 segment into the global ruleset, as indicated by the green rectangular box in the image 4.8 below.

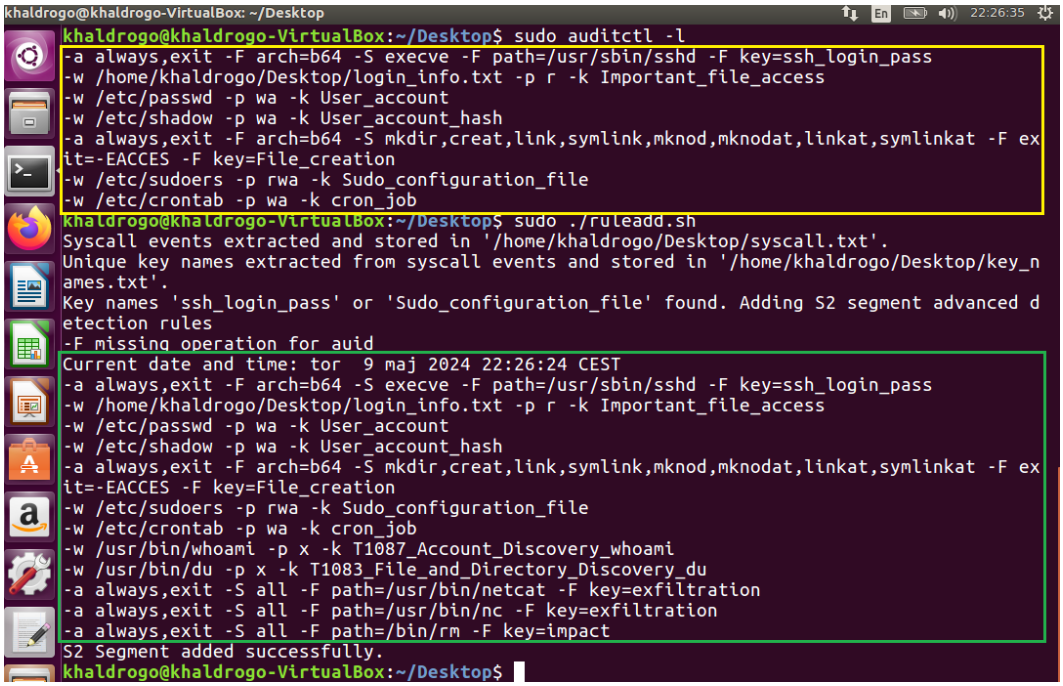

Figure 4.8: S2 Segment Addition

The segment S2 was added to the global ruleset at 22:26:24 on May 9, 2024.

Step 4: System Information Discovery - After a 10-second delay following the earlier privilege escalation, the attacker executed the 'whoami' command to determine the username of the current user. AuditD triggered an alert based on the ruleset at 22:26:30 on May 9, 2024, as depicted in Figure 4.9.

Figure 4.9: Current User Information

After waiting for 5 seconds, the attacker executed the 'du' command to gather information about disk usage on the system and identify sensitive information. This action took place at 22:26:35, as depicted in Figure 4.10.

The logic used to detect malware activities prompted the inclusion of the S2 segment into the global ruleset. This selection logic effectively pinpointed the system discovery information targeted by the malware by adding the appropriate segment.

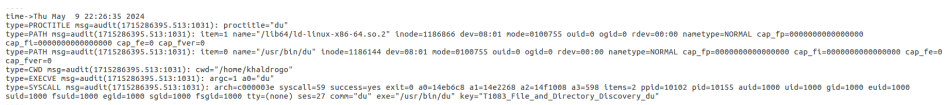

Figure 4.10: Disk Usage

This systematic approach optimizes threat detection and efficiency by dynamically incorporating suitable segments based on triggered alerts. Furthermore, the timestamps of the attack delays were logged in test case 2 within the AuditD records, confirming the successful detection of a multi-stage attack on the target machine.

4.2. Performance Evaluation of the Target Machine

To evaluate system performance, the system execution time was analyzed across three conditions: a Linux system without AuditD, one with AuditD and the global ruleset, and another with AuditD and the full ruleset.

In the absence of AuditD, the average execution time for the syscall getppid in the Linux system was 107.32 ns, with a median of 170.00 ns. The minimum execution time was 102 ns, while the maximum was 1466 ns. Given the wide range, the y-axis in the figure below 4.11 was considered from 1 to 250 ns.

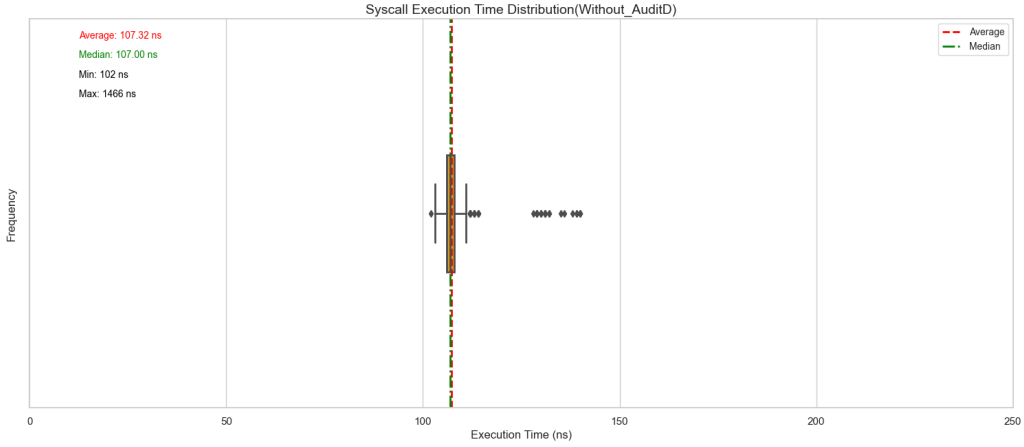

Figure 4.11: Without AuditD

With AuditD installed on the target machine and configured with the global ruleset, the average system call execution recorded time was 198.07 ns, with a median of 196.00 ns. The minimum execution time was 186 ns, and the maximum was 1997 ns, as depicted in figure 4.12.

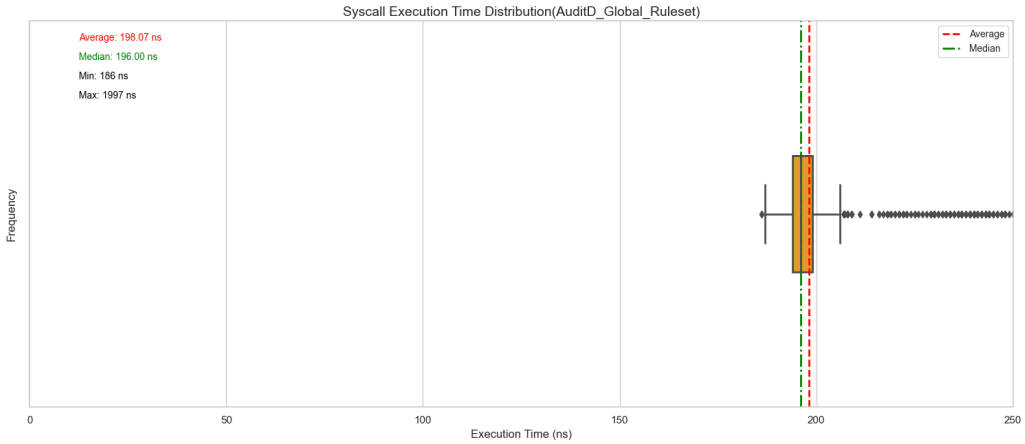

Figure 4.12: AuditD with Global Ruleset

For AuditD with the full ruleset configured on the target machine, the average system execution time was 351.69 ns, with a median of 315.0 ns. The minimum time recorded was 298 ns, while the maximum was 3891 ns. Given the wide range, the y-axis for the execution time was considered from 1 to 500 ns, as illustrated in the figure below 4.13.

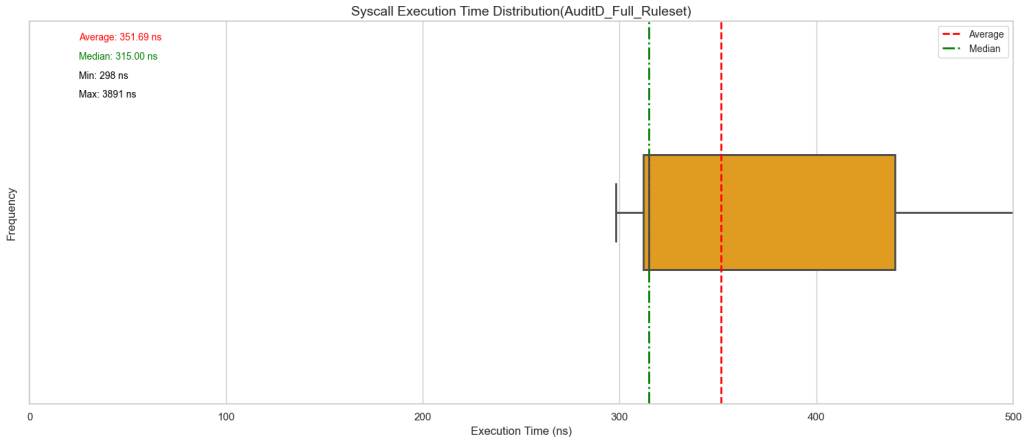

Figure 4.13: AuditD with Full Ruleset

The statistics for all three conditions are depicted in the figure below 4.14.

Figure 4.14: Comparison of Three Conditions

When evaluating system performance under normal runtime conditions without rule triggering, it's important to note that the system without AuditD installed serves as the baseline. In this scenario, the average system execution time was 107.32 ns, with a median of 107.00 ns.

Upon installing AuditD with the global ruleset, there was an 84.56% increase in system execution time, with the median increasing by 83.18%. This indicates a significant increase in the time taken by the system to process syscalls. In comparison, when AuditD with the full ruleset was installed, the system execution time increased by 227.70%, with the median increasing by 194.39%.

Furthermore, when comparing system call execution with the global ruleset and the full ruleset, there was a 143.14% increase. This substantial increase further emphasizes the impact of utilizing different rule configurations on system performance.

The system performance under open syscall rule triggering was evaluated under two conditions: one triggering the rule with the full ruleset and the other triggering it with the global ruleset. The data presented below shows execution times in microseconds.

When triggering the rule with the global ruleset, the average execution time of the target system was 169.02 μs, with a median time of 11.26 μs. The minimum execution time recorded was 9.94 μs, while the maximum was 237636.15 μs, as depicted in Figure 4.15. Considering the wide range of values on the y-axis, the data representation was adjusted to range from 1 to 180 μs.

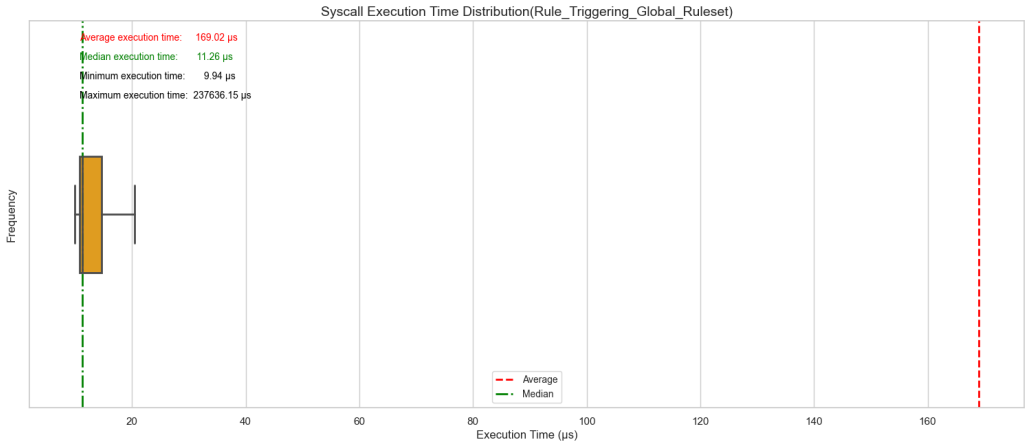

Figure 4.15: Rule Triggering with Global Ruleset

With the full ruleset configured on the target system and open syscall rule triggering, the average system execution time was 176.15 μs, with a median of 11.42 μs. The minimum execution time recorded was 10.05 μs, and the maximum was 254675.45 μs, shown in Figure 4.16. The presented data showed a wide range of execution times on the y-axis. To effectively display the data, the range was considered from 1 to 180 μs.

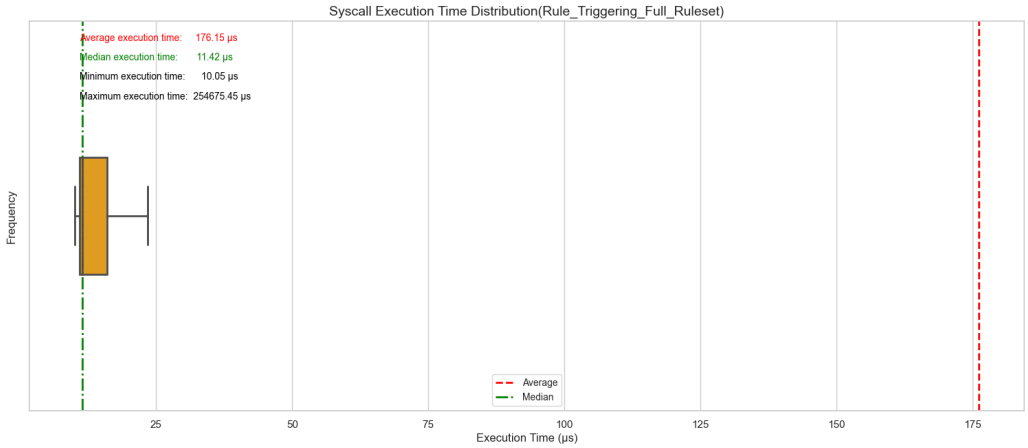

Figure 4.16: Rule Triggering with Full Ruleset

A comparison of both scenarios was presented in Figure 4.17.

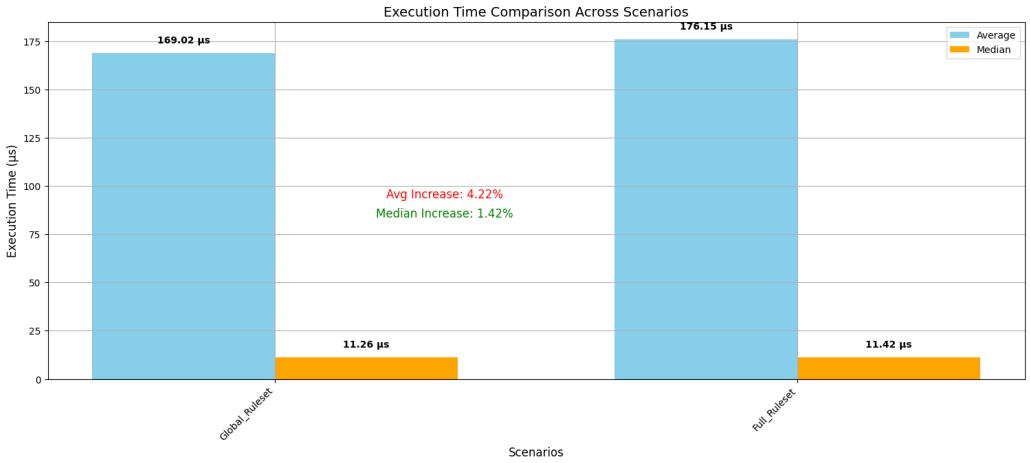

Figure 4.17: Comparison of Scenarios - Rule Triggering

When examining system performance triggered by alerts, which signifies rule activation based on system actions, note distinct differences between configurations. With the global ruleset enabled, compared to the full ruleset, the average system call execution time increases by 4.22%, with the median rising by 1.42%. This suggests that enabling the full ruleset results in longer processing times for the system.

5. Discussion

This discussion covers the methodology, implementation, results, validity, and reliability of the experiments, research contribution, potential improvements, as well as limitations, and ethical and societal considerations.

5.1. Method, Implementation, and Results

5.1.1 Segmentation and Dynamic Expansion Technique

The segmentation and dynamic expansion technique streamline threat detection by categorizing rulesets into initial, choke point, and advanced detection, aligned with the MITRE Attack framework which provides a granular view of effective threat detection. This structured approach enhances system security, allowing for targeted threat detection and optimizing resource utilization. Additionally, the method reduces overhead by combining initial and choke point detection rules into a global ruleset and dynamically adds the advanced detection rules based on attacker actions.

Despite its structured approach, categorizing rulesets based on MITRE tactics may overlook the possibility of attackers using tactics or techniques in the early stages instead of advanced stages of detection. Since attacker behavior is dynamic in nature, this limitation can result in missed threats during early detection stages, underscoring the need for ongoing refinement and validation of the rule categorization process. Moreover, the segmentation process may introduce complexity and overhead, particularly in scenarios with numerous rulesets. This complexity can lead to increased management efforts and resource consumption, potentially resulting in rule redundancy across segments. Additionally, maintaining rule consistency and coherence in dynamically evolving environments poses challenges, highlighting the importance of regular audits and reviews to ensure accuracy and minimize false positives.

To address these challenges, future research could focus on refining the segmentation process through automated tools or algorithms to minimize complexity while maintaining effectiveness. Rather than pinpointing specific choke points for each attacker's objective, it is more efficient to establish more generalized choke points to effectively manage the overall strategy. Moreover, implementing robust auditing mechanisms and streamlining rule management processes can further enhance system performance and reliability. Continuous monitoring and validation of rule categorization logic are essential to ensure comprehensive threat coverage and reduce the risk of missed detections. Additionally, implementing robust auditing mechanisms and streamlining rule management processes can further enhance system performance and reliability.

5.1.2 Selection Logic and Adaptability:

The selection logic enables systematic evaluation of rule combinations to identify suitable segments for detection based on attacker actions. This adaptability allows orga-

nizations to tailor detection capabilities to specific threat scenarios, enhancing overall resilience against evolving cyber threats. However, the selection logic may introduce additional complexity in rule management and configuration, especially as the number of rules and segments grows. Maintaining consistency and coherence in rule categorization and selection logic becomes increasingly challenging under these circumstances. Furthermore, the delay observed in adding the segment suggests a potential impact on detecting and addressing emerging threats.

To address these challenges, organizations can focus on continued refinement of the selection logic. Coupling it with automation and machine learning techniques can help streamline rule management processes and improve adaptability. Regular updates to the selection logic could accommodate changes, such as malware behavior, by incorporating new combinations and rules into appropriate segments.

Furthermore, this approach can be extended to test HIDS in other operating systems apart from Linux distributions. Integrating threat intelligence feeds into the segmentation and dynamic expansion technique can further enhance the detection capabilities. By leveraging AI-driven approaches, organizations can achieve more efficient rule management without sacrificing accuracy or coherence.

5.1.3 Impact on System Performance:

Evaluation of system performance under different configurations provides valuable insights into the impact of rulesets on system resources. By quantifying the performance differences between configurations, organizations can make informed decisions regarding rule optimization and resource allocation. However, performance evaluation alone may not fully capture the broader implications of rule complexity on overall system security. While performance improvements are desirable, they should not come at the expense of effective threat detection and mitigation.

Future research could explore holistic metrics for evaluating system security that consider both performance and effectiveness in threat detection. By balancing these factors, organizations can optimize system configurations to maximize both performance and security posture. Additionally, future research can review the placement of the rule and assess its effectiveness in providing comprehensive threat coverage while also considering its impact on system performance.

5.1.4 Validity and Reliability of the Experiments

Utilizing open-source software like the GitHub repository (bfuzzy 2024) ensured that research methods were accessible and reproducible by others, promoting transparency and accountability. The test cases were carefully chosen to cover potential attack vectors and scenarios, reflecting real-world situations that an intrusion detection system might face. This included simulating both common and sophisticated attack techniques as outlined in the MITRE ATT&CK framework. The experimental environment was designed to closely mimic the actual configurations of embedded systems and network setups. This included using typical hardware and software stacks to ensure that the results would apply to real-world embedded systems. Automated malicious scripts were utilized to minimize human error and ensure that procedures were precisely followed

in each run. Data from each experiment were meticulously recorded and stored, and statistical methods were employed to analyze the data, ensuring consistent results. To further enhance validity, the system's performance was tested under three different ruleset configurations: without AuditD, with AuditD and a global ruleset, and with AuditD and a full ruleset. This approach allowed for a comprehensive assessment of how different ruleset complexities impact system performance. Overall, these measures contribute to the validity of the experiments by ensuring that they accurately represent real-world scenarios and are conducted ethically and transparently.

The reliability of the experiments is upheld by several factors. Firstly, the automated malicious scripts minimize human error and ensure consistent procedural execution. Additionally, the availability of the segmentation selection logic algorithm and the comprehensive script allows for the replication of segment addition. Moreover, the thorough recording and storage of data, along with the application of statistical methods such as averages and medians for analysis, further contribute to the reliability of the findings by enabling replication and verification. Overall, these measures ensure that the experiments are reliable and can be trusted to provide accurate and consistent findings.

5.1.5 Research Contribution

The study conducted by Vermeer, Eeten, and Gañán (2022) identified common practices in IDS management, such as creating proprietary rules to address client-specific threats and updating these rules more frequently than commercial rulesets. However, this research goes a step further by implementing these practices and emphasizing client-specific threats through the development of a threat model and identifying potential threats before considering commercial rulesets. Unlike traditional methods that rely on static rule management, this novel approach categorizes the rulesets into initial detection, choke point, and advanced detection to provide a more granular and customized view of threat detection. Vermeer also suggested reducing false positive incidents by updating rules that trigger corresponding alerts. In contrast, this study further develops and demonstrates this idea in practice by dynamically incorporating segments based on attacker actions, thereby effectively mitigating false positives.

The research conducted by Ji et al. (2015) focused on analyzing overhead and evaluating various approaches to host-based bot detection. Their study proposed several alternatives, including enhancing the correlation engine by reducing time complexity through improved indexing to minimize unrelated matches. This research implements their suggestions through segmentation and dynamic expansion techniques. In this approach, only the global ruleset is matched against all events instead of the full ruleset. Based on attacker behavior, the appropriate segment is dynamically added to identify potential threats. Consequently, this strategy significantly reduces time complexity, specifically the system time required to process events against the ruleset. The research methodology and experimental results demonstrate that this approach aligns with the recommendations made by Ji et al. (2015), offering a practical way to reduce time complexity in host-based intrusion detection systems.

5.1.6 Limitations

While the methodology effectively identifies two-step or multi-stage attacks, it may not adequately cover a broad range of activities. This poses a risk of undetected intrusion activity if subsequent actions following system access are not accounted for by the rules. Additionally, there exists a detection gap when an attacker gains system access and executes actions not explicitly addressed by the rules in the particular segment. For instance, the segmentation and dynamic expansion technique's effectiveness in detecting multi-stage attacks might be constrained by the choke points identified through data-driven studies. While these choke points provide valuable insights, they may not encompass the entire spectrum of potential attack scenarios, leading to gaps in threat coverage. Moreover, reliance on simulated attack scenarios may not fully capture the complexity and variability of real-world cyber threats. Furthermore, the experimental setup's lack of compatibility with an embedded system scenario, as it was hosted on a local machine using virtualization, presents an additional limitation. Although test cases based on the designed threat model offer valuable insights into the performance of IDS rulesets under controlled conditions, they may not precisely mirror the challenges encountered in live environments with evolving threat landscapes.

5.2. Ethical and Societal Aspects

When conducting experiments involving attacks on target machines in controlled environments, it's essential to address ethical considerations. Malicious actors could potentially exploit research findings to harm legitimate services, emphasizing the need to ensure that researchers aim to manage extensive IDS rulesets while minimizing performance overhead on embedded systems. Ethical concerns arise due to the potential consequences of false positives or false negatives. False alarms can disrupt operations unnecessarily, while missed detections can lead to security breaches and data compromises.

Furthermore, the societal impact of IDS technologies plays a crucial role in cybersecurity defense strategies and digital resilience. Innovative approaches like segmentation and dynamic expansion represent proactive measures against evolving cyber threats, bolstering organizational security and safeguarding critical infrastructures. By enhancing threat detection capabilities and mitigating security risks, IDS technologies contribute to overall societal well-being by fostering trust in digital systems and enabling secure digital interactions.

6. Conclusion

The management of extensive rulesets within Host Intrusion Detection Systems (HIDS) presents a critical challenge in modern cybersecurity. This research proposes a novel approach to address this challenge through the segmentation and dynamic expansion technique. By categorizing rulesets into initial detection, choke point detection, and advanced detection based on the MITRE Att&ck framework, the methodology aims to streamline threat detection and optimize resource utilization.

The results of the study demonstrate the effectiveness of this approach in reducing the overhead associated with extensive rulesets. By combining initial and choke point detection rules into a global ruleset and dynamically adding advanced detection rules based on attacker actions, the methodology minimizes the need for exhaustive rule matching, leading to improved efficiency and resource utilization.

However, there are limitations and areas for improvement to consider. While the structured approach enhances threat detection, it may not capture the full spectrum of dynamic attacker techniques and behaviors. Moreover, the segmentation process introduces complexity and overhead, particularly in scenarios with numerous rulesets. Maintaining rule consistency and coherence in dynamically evolving environments also poses challenges.

In conclusion, the segmentation and dynamic expansion technique offers a promising approach to managing extensive rulesets within HIDS, enhancing system security and threat detection capabilities. By addressing the challenges associated with rule management and optimization, this methodology lays the foundation for more efficient and effective cybersecurity practices in the face of evolving threats.

6.1. Future work

The future research could focus on refining the segmentation process through automated tools or algorithms to minimize complexity while maintaining effectiveness. Continuous monitoring and validation of rule categorization logic are essential to ensure comprehensive threat coverage and reduce the risk of missed detections. Additionally, implementing robust auditing mechanisms and streamlining rule management processes can further enhance system performance and reliability.

Expansion to Other FOSS: Extending the segmentation and dynamic expansion techniques to different Free Open-Source Software (FOSS) platforms and conducting comprehensive performance evaluations. By applying the methodology to various FOSS environments, researchers can assess its effectiveness and threat coverage across different software ecosystems through data-driven studies.

Performance Evaluation with Rule Segments: Future work could involve evaluating the performance of selecting rule segments in conjunction with threat detection. This would entail analyzing the time difference in fetching rule segments after detecting choke points. Assessing the impact of fetching speed on AuditD performance, particularly in scenarios where AuditD might be under attack, would be crucial. This evaluation could provide insights into enhancing system responsiveness and resilience in the face of security threats.

Enhancement of Selection Logic: Continuing to enhance the selection logic to address limitations and adapt to evolving threat landscapes. By refining the selection logic, researchers can improve the methodology's ability to accurately identify suitable segments for detection based on attacker actions, thereby enhancing overall threat detection capabilities.

Enhancing Rule Positioning Efficiency: Exploring the effectiveness of rule positioning in accordance with alert triggering to prioritize rules and improve detection efficiency. Conducting experiments to analyze the impact of rule positioning within the ruleset on detection efficacy could lead to optimizations that enhance the overall performance of the detection system.

Bibliography

Aparicio-Navarro, Franciso J et al. (2018). "Multi-stage attack detection using contextual information". In: *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, pp. 1–9.

ArchWiki (2023). *Audit framework*. https://wiki.archlinux.org/title/Audit_framework. ArchWiki. Retrieved from the ArchWiki website.

Arshad, Junaid et al. (2019). "An intrusion detection framework for energy constrained IoT devices". In: *Mechanical Systems and Signal Processing*, p. 106436. DOI: https://doi.org/10.1016/j.ymssp.2019.106436.

Asad, Hassan and Ilir Gashi (2018). "Diversity in Open Source Intrusion Detection Systems". In: *Lecture Notes in Computer Science*, pp. 267–281. DOI: https://doi.org/10.1007/978-3-319-99130-6_18.

bfuzzy (Apr. 2024). *bfuzzy/auditd-attack*. https://github.com/bfuzzy/auditd-attack. GitHub repository. Retrieved from GitHub website.

Bhatt, Parth, Edgar Toshiro Yano, and Per Gustavsson (2014). "Towards a framework to detect multi-stage advanced persistent threats attacks". In: *2014 IEEE 8th international symposium on service oriented system engineering*. IEEE, pp. 390–395.

Bures, Michal et al. (2021). "Review of Specific Features and Challenges in the Current Internet of Things Systems Impacting their Security and Reliability". In: *arXiv (Cornell University)*. DOI: https://doi.org/10.48550/arxiv.2101.02631.

Cole, Christian (2024). *IoT 2023 in review: The 10 most relevant IoT developments of the year*. Online. Accessed 13 Mar. 2024. IoT Analytics. URL: https://iot-analytics.com/iot-2023-in-review/.

Cozzi, Emanuele et al. (2018). "Understanding linux malware". In: *2018 IEEE symposium on security and privacy (SP)*. IEEE, pp. 161–175.

Dawson, Kara and Richard Ferdig (2006). "Commentary: Expanding Notions of Acceptable Research Evidence in Educational Technology: A Response to Schrum et al." In: *Contemporary Issues in Technology and Teacher Education* 6.1. Accessed 24 Mar. 2024, pp. 133–142. URL: https://www.learntechlib.org/p/21822/.

GeeksforGeeks (May 2021). *Linux Directory Structure*. https://www.geeksforgeeks.org/linux-directory-structure/. GeeksforGeeks. Retrieved from GeeksforGeeks website.

Google Security Operations (2024). *Overview of Linux Threats category*. https://cloud.google.com/chronicle/docs/detection/linux-threats-category. Google Cloud. Retrieved April 29, 2024, from Google Cloud website.

Hashem, Yousef and Elmedin Zildzic (2022). *Endpoint Intrusion Detection and Response Agents in Embedded RAN Products: A suitability and performance evaluation*.

Ji, Yang et al. (2015). "Overhead Analysis and Evaluation of Approaches to Host-Based Bot Detection". In: *International Journal of Distributed Sensor Networks* 11.5, pp. 524627–524627. DOI: https://doi.org/10.1155/2015/524627.

Levy, Y. and T. J. Ellis (2006). "A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research". In: *Informing Science:*

The International Journal of an Emerging Transdiscipline 9.1, pp. 181–212. DOI: 10.28945/479. URL: https://doi.org/10.28945/479.

Linux — getppid (2024). *getppid - Linux man page*. Linux Man Pages. URL: https://linux.die.net/man/2/getppid.

Linux — open (2024). *open(2) - Linux manual page*. Www.man7.org. https://www.man7.org/linux/man-pages/man2/open.2.html.

linux-audit/audit-kernel (Mar. 2024). *audit-kernel*. https://github.com/linux-audit/audit-kernel. GitHub. Retrieved from GitHub repository.

Liu, Qiang, Veit Hagenmeyer, and Hans-Berndt Keller (2021). "A Review of Rule Learning-Based Intrusion Detection Systems and Their Prospects in Smart Grids". In: 9, pp. 57542–57564. DOI: https://doi.org/10.1109/access.2021.3071263.

Mathew, Sunu et al. (2005). "Real-time multistage attack awareness through enhanced intrusion alert clustering". In: *MILCOM 2005-2005 IEEE Military Communications Conference*. IEEE, pp. 1801–1806.

Microsoft (Aug. 2022). *Threats - Microsoft Threat Modeling Tool - Azure*. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats. Microsoft Azure. Retrieved from Microsoft Learn website.

MITRE (2024). *MITRE ATT&CK™*. https://attack.mitre.org/. MITRE. Retrieved from MITRE website.

MITRE - Credential Dumping (2012). *Credential Dumping - Enterprise — MITRE ATT&CK™*. https://attack.mitre.org/techniques/T1003/. MITRE. Retrieved from MITRE website.

MITRE — Data Source (2024). *File, Data Source DS0022 — MITRE ATT&CK®*. https://attack.mitre.org/datasources/DS0022/. MITRE. Retrieved April 29, 2024, from MITRE website.

MITRE ATT&CK —General Information (2024). *MITRE ATT&CK – Get Started*. https://attack.mitre.org/resources/. MITRE. Retrieved April 28, 2024, from MITRE website.

MITRE ATT&CK —Tactics (2015). *Tactics - Enterprise — MITRE ATT&CK™*. https://attack.mitre.org/tactics/enterprise/. MITRE. Retrieved from MITRE website.

MITRE ATT&CK —Techniques (2023). *Techniques - Enterprise — MITRE ATT&CK®*. https://attack.mitre.org/techniques/enterprise/. MITRE. Retrieved from MITRE website.

MITRE ATT&CK Enterprise (2024). *Matrix - Enterprise — MITRE ATT&CK®*. https://attack.mitre.org/matrices/enterprise/linux/. MITRE. Retrieved from MITRE website.

Naik, Nitin et al. (2022). "Comparing attack models for it systems: Lockheed martin's cyber kill chain, mitre att&ck framework and diamond model". In: *2022 IEEE International Symposium on Systems Engineering (ISSE)*. IEEE, pp. 1–7.

Okoli, Chitu (2015). "A Guide to Conducting a Standalone Systematic Literature Review". In: *Communications of the Association for Information Systems* 37.1.

OWASP Foundation —Pythonic Threat Modeling (2024). *Pythonic Threat Modeling*. https://owasp.org/www-project-developer-guide/release/design/threat_modeling/pytm/. OWASP Foundation. Retrieved April 28, 2024, from OWASP website.

OWASP Foundation —Threat Modeling (2022). *The OWASP Foundation Threat Modeling*. URL: https://owasp.org/www-community/Threat_Modeling.

Rajesh, P et al. (2022). "Analysis of cyber threat detection and emulation using mitre attack framework". In: *2022 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*. IEEE, pp. 4–12.

Sen, Sumanta (2006). *Performance Characterization Improvement of Snort as an IDS*. Accessed 13 Mar. 2024. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=019581c7780f32ad7b79440e3c28a6a94cb0f247.

Sen, Sumanta, Jaehoon Koo, and Saurabh Bagchi (2018). "TRIFECTA: Security, Energy Efficiency, and Communication Capacity Comparison for Wireless IoT Devices". In: *IEEE Internet Computing* 22.1, pp. 74–81. DOI: https://doi.org/10.1109/mic.2018.011581520.

Soliman, Khaled Sayed, Mohammed A. Sobh, and Ahmed M. Bahaa-Eldin (2021). "Survey of Machine Learning HIDS Techniques". In: DOI: https://doi.org/10.1109/icces54031.2021.9686138.

Takey, Yuvraj Sanjayrao et al. (2021). "Real time early multi stage attack detection". In: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. IEEE, pp. 283–290.

Tarandach, Izar (Apr. 2024). *izar/pytm*. https://github.com/izar/pytm. GitHub. Retrieved from GitHub repository.

Thongkanchorn, Kanokorn, Siriwat Ngamsuriyaroj, and Vorapong Visoottiviseth (2013). "Evaluation studies of three intrusion detection systems under various attacks and rule sets". In: *IEEE Xplore*. DOI: https://doi.org/10.1109/TENCON.2013.6718975.

Turner, Chris et al. (2016). "A Rule Status Monitoring Algorithm for Rule-Based Intrusion Detection and Prevention Systems". In: *Procedia Computer Science* 95, pp. 361–368. DOI: https://doi.org/10.1016/j.procs.2016.09.346.

Vasilomanolakis, Emmanouil et al. (2016). "Multi-stage attack detection and signature generation with ICS honeypots". In: *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*. IEEE, pp. 1227–1232.

Vermeer, Maarten, Michel van Eeten, and Carlos Gañán (2022). "Ruling the Rules". In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. DOI: https://doi.org/10.1145/3488932.3517412.

White, J. A., T. Fitzsimmons, and J. Matthews (2013). "Quantitative analysis of intrusion detection systems: Snort and Suricata". In: *Proceedings of SPIE*. DOI: https://doi.org/10.1117/12.2015616.

Wikipedia (Mar. 2019). *Transmission Control Protocol*. Wikipedia. Accessed 13 Mar. 2024. Wikimedia Foundation. URL: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

– (Feb. 2020). *System call*. https://en.wikipedia.org/wiki/System_call.

– (Apr. 2024). *STRIDE model*. https://en.wikipedia.org/wiki/STRIDE_model. Wikipedia. Retrieved from Wikipedia website.

Zarpelão, Bruno B. et al. (2017). "A survey of intrusion detection in Internet of Things". In: *Journal of Network and Computer Applications* 84, pp. 25–37. DOI: 10.1016/j.jnca.2017.02.009. URL: https://doi.org/10.1016/j.jnca.2017.02.009.

A. Appendices

A.1. Pytm Source Code

```
from pytm.pytm import TM, Server, Datastore, Dataflow,
   Actor

# Create a threat model
tm = TM("Linux System Threat Model")
tm.description = "Threat model for a Linux system with
   basic components"

# Define actors
external_user = Actor("External User", is_external=True)

# Define components
linux_system = Server("Linux System")
browser = Server("Browser")

# Define data stores
important_files = Datastore("Important Files")

# Define data flows
external_access_to_linux = Dataflow(external_user,
   linux_system, "Access Linux System (Direct)")
browser_to_linux = Dataflow(browser, linux_system, "Access
   Linux System (via Browser)")
linux_to_files = Dataflow(linux_system, important_files, "
   Access Important Files")

# New data flow: SSH login to Linux system
ssh_login_to_linux = Dataflow(external_user, linux_system,
   "Login to Linux System via SSH")

# Generate and print threat model
tm.process()
tm_json = tm.json()
print(tm_json)
```

Code extraction A.1: pytm source code

A.2. Adversarial Bash Script - Test Case 1

```
#!/bin/bash
# Variables for remote SSH login
REMOTE_HOST="192.168.100.7"
REMOTE_USER="khaldrogo"
REMOTE_PASS="admin"
ROOT_PASS="admin"
SSH_PORT="22"

# Source file path (Sensitive files in the system)
SOURCE_FILE_PATH="/home/khaldrogo/Desktop/login_info.txt"

# Attempt to login to remote machine using sshpass
echo "Attempting SSH login to $REMOTE_HOST..."
sshpass -p "$REMOTE_PASS" ssh -p "$SSH_PORT" "$REMOTE_USER"
   @"$REMOTE_HOST"<< EOF
    sleep 5

    # Accessing sensitive file by opening it
    echo "Opening the file using cat /login_info.txt"
    cat "$SOURCE_FILE_PATH"
    sleep 10

    # Retrieve system information: user account information
    echo "System information:"
    id
    sleep 5

    # Data Encoding
    base64
    sleep 10

    # Exit the SSH session
    exit
EOF

echo "Script completed."
```

Code extraction A.2: Test Case 1

A.3. Adversarial Bash Script - Test Case 2

```
#!/bin/bash
# Variables for remote SSH login
REMOTE_HOST="192.168.100.7"
REMOTE_USER="khaldrogo"
REMOTE_PASS="admin"
ROOT_PASS="admin"
SSH_PORT="22"

# Attempt to login to remote machine using sshpass
echo "Attempting SSH login to $REMOTE_HOST..."
sshpass -p "$REMOTE_PASS" ssh -p "$SSH_PORT" "$REMOTE_USER"
   @"$REMOTE_HOST"<< EOF

    sleep 5
    # Try to switch to root user using sudo and opening
       sudo configuration file
    echo "Trying to switch to root user..."
    echo "$ROOT_PASS" | sudo -S cat /etc/sudoers
    if ! echo "$ROOT_PASS" | sudo -S cat /etc/sudoers; then
         echo "Failed to switch to root user. Check for
            errors."
    fi

    sleep  10
    # Retrieve system information: whoami
    echo "system information:"
    whoami
    sleep 5

    echo "listing all the directories:"
    du
    sleep 10


    # Exit the SSH session
    exit
EOF
```

Code extraction A.3: Test Case 2

A.4. AuditD Full Ruleset

```
#Initial detection ruleset
-a always,exit -F arch=b64 -S execve -F path=/usr/sbin/sshd
    -F key=ssh_login_pass

#Chokepoint detection ruleset for selecting S1 segment
-w /home/khaldrogo/Desktop/login_info.txt -p r -k
   Important_file_access
-w /etc/passwd -p wa -k User_account
-w /etc/shadow -p wa -k User_account_hash
-a always,exit -F arch=b64 -S mkdir,creat,link,symlink,
   mknod,mknodat,linkat,symlinkat -F exit=-EACCES -k
   File_creation

#Chokepoint detection ruleset for selecting S2 segment
-w /etc/sudoers -p rwa -k Sudo_configuration_file
-w /etc/crontab -p wa -k cron_job

#Segment 1 - Advance detection ruleset
-w /usr/bin/id -p x -k T1087_Account_Discovery_id
-w /usr/bin/base64 -p x -k susp_activity
-w /usr/bin/find -p x -k T1083_File_and_Directory_Discovery
-a always,exit -F arch=b64 -S kill -F key=impactcle
-w /usr/bin/ftp -p x -k T1105_remote_file_copy

#Segment 2 - Advance detection ruleset
-w /usr/bin/whoami -p x -k T1087_Account_Discovery_whoami
-w /usr/bin/du -p x -k
   T1083_File_and_Directory_Discovery_du
-a always,exit -F path=/usr/bin/netcat -F key=exfiltration
-a always,exit -F path=/usr/bin/nc -k key=exfiltration
-a always,exit -F path=/bin/rm -F key=impact
```

Code extraction A.4: Full Detection Ruleset

A.5. Segment Selection Logic Script

```
#!/bin/bash

# Define the input log file path
LOG_FILE="/home/khaldrogo/Desktop/log.txt"

# Define the output file path for storing unique key names
OUTPUT_FILE="/home/khaldrogo/Desktop/key_names.txt"

# Define the output file path for storing syscall entries
SYSCALL_FILE="/home/khaldrogo/Desktop/syscall.txt"

# Check for sudo privileges
if sudo -n true; then
    # Append recent events from the last 10 minutes to log.
       txt
    sudo ausearch --start recent >> "$LOG_FILE"

    # Filter log entries to extract only syscall events and
        save to syscall.txt
    if [ -f "$LOG_FILE" ]; then
        grep 'type=SYSCALL' "$LOG_FILE" > "$SYSCALL_FILE"
        echo "Syscall events extracted and stored in '
           $SYSCALL_FILE'."

        # Extract key names from syscall.txt and store in
           OUTPUT_FILE
        grep -o 'key="[^"]*"' "$SYSCALL_FILE" | sed 's/key
           ="//; s/"$//' | sort -u > "$OUTPUT_FILE"
        echo "Unique key names extracted from syscall
           events and stored in '$OUTPUT_FILE'."

        # Check if specific key names exist in OUTPUT_FILE
        if grep -q 'ssh_login_pass' "$OUTPUT_FILE" && grep
           -q 'Important_file_access' "$OUTPUT_FILE"; then
            echo "Key names 'ssh_login_pass' or '
               Important_file_access' found. Adding S1
               segment advanced detection rules"

            # Add audit rules using sudo auditctl for
               specific key names (example rules)
            sudo auditctl -w /usr/bin/id -p x -k
               T1087_Account_Discovery_id
            sudo auditctl -w /usr/bin/base64 -p x -k
               susp_activity
            sudo auditctl -w /usr/bin/find -p x -k
```

```
            T1083_File_and_Directory_Discovery
        sudo auditctl -a always,exit -F arch=b64 -S
            kill -k impactcle
        sudo auditctl -w /usr/bin/ftp -p x -k
            T1105_remote_file_copy

        # Add more audit rules for combination 1 as
            needed
        echo "Current date and time: $(date)"
        sudo auditctl -l

        echo "S1 Segment added successfully."
    elif grep -q 'ssh_login_pass' "$OUTPUT_FILE" &&
       grep -q 'Sudo_configuration_file' "$OUTPUT_FILE"
       ; then
        echo "Key names 'ssh_login_pass' or '
            Sudo_configuration_file' found. Adding S2
            segment advanced detection rules"

        # Add audit rules using sudo auditctl for
            specific key names (example rules)
        sudo auditctl -w /usr/bin/whoami -p x -k
            T1087_Account_Discovery_whoami
        sudo auditctl -w /usr/bin/du -p x -k
            T1083_File_and_Directory_Discovery_du
        sudo auditctl -a always,exit -F path=/usr/bin/
            netcat -k exfiltration
        sudo auditctl -a always,exit -F path=/usr/bin/
            nc -k exfiltration
        sudo auditctl -a always,exit -F path=/bin/rm -k
             impact
        sudo auditctl -a always,exit -F arch=b64 -S
            connect -F auid>=1000 -F auid!=4294967295 -F
             auid!=obj_uid -F key=known_c2_server -k
            T1043_Commonly_Used_Port
        # Add more audit rules for combination 2 as
            needed
        echo "Current date and time: $(date)"
        sudo auditctl -l

        echo "S2 Segment added successfully."
    else
        echo "No specific key names found in '
            $OUTPUT_FILE'."
    fi
else
    echo "Error: Log file '$LOG_FILE' not found."
```

```
        exit 1
    fi
else
    echo "Error: You do not have sudo privileges or sudo
       access has expired."
    exit 1
fi
```

Code extraction A.5: Segment selection: ./ruleadd.sh

A.6. System Execution Time Script - getppid

```
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <stdint.h>

int main() {
    struct timespec start, end;
    FILE *logfile = fopen("getppid_times.csv", "w");
    if (logfile == NULL) {
        perror("fopen");
        return 1;
    }

    // Write CSV header
    fprintf(logfile, "call_index,execution_time_ns\n");

    for (int i = 0; i < 10000; ++i) {
        // Start timing
        clock_gettime(CLOCK_MONOTONIC, &start);

        // Call getppid
        getppid();

        // Stop timing
        clock_gettime(CLOCK_MONOTONIC, &end);

        // Calculate the elapsed time in nanoseconds
        int64_t elapsed_ns = (end.tv_sec - start.tv_sec) *
           1000000000LL + (end.tv_nsec - start.tv_nsec);

        // Log the index and execution time to the CSV file
        fprintf(logfile, "%d,%ld\n", i + 1, elapsed_ns);
    }

    fclose(logfile);
    printf("Execution times logged to getppid_times.csv\n")
       ;

    return 0;
}
```

Code extraction A.6: getppid script

A.7. System Execution Time Script - open

```
#include <stdio.h>
#include <time.h>
#include <fcntl.h>
#include <stdint.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>

int main() {
    struct timespec start, end;
    const char *filename = "/home/khaldrogo/Desktop/test.
       txt";
    int fd;
    FILE *logfile = fopen("open_times.csv", "w");

    if (logfile == NULL) {
        perror("fopen");
        return 1;
    }

    fprintf(logfile, "call_index,execution_time_ns\n");

    for (int i = 0; i < 10000; ++i) {
        clock_gettime(CLOCK_MONOTONIC, &start);

        // Open the file
        fd = open(filename, O_CREAT | O_WRONLY | O_TRUNC,
           0644);
        if (fd == -1) {
            fprintf(stderr, "Error opening file: %s\n",
               strerror(errno));
            fclose(logfile);
            return 1;
        }

        // Close the file
        if (close(fd) == -1) {
            fprintf(stderr, "Error closing file: %s\n",
               strerror(errno));
            fclose(logfile);
            return 1;
        }

        clock_gettime(CLOCK_MONOTONIC, &end);
```

```
        // Calculate execution time
        int64_t elapsed_ns = (end.tv_sec - start.tv_sec) *
           1000000000LL + (end.tv_nsec - start.tv_nsec);

        // Log the index and execution time to the CSV file
        fprintf(logfile, "%d,%ld\n", i + 1, elapsed_ns);
    }

    fclose(logfile);
    printf("File open/close times logged to open_times.csv\
       n");

    return 0;
}
```

Code extraction A.7: open syscall script