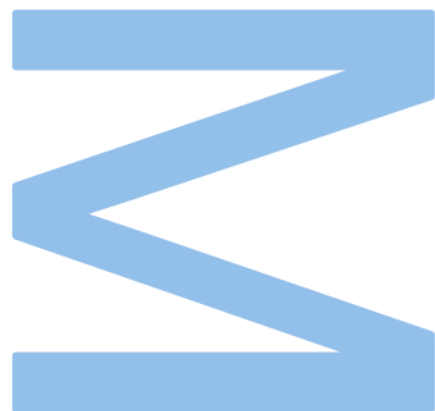
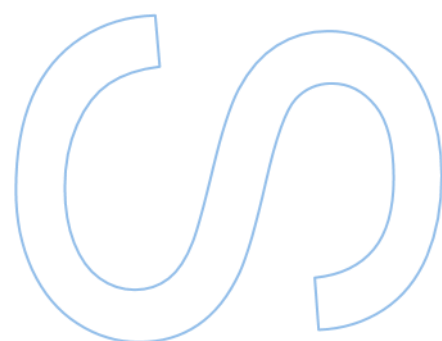


Syscall Security Components



Henrique Pessoa Léo

Master's Degree in Information Security
Department of Computer Science
Faculty of Sciences of the University of Porto
2024



Syscall Security Components

Henrique Pessôa Léo

Dissertation Report carried out as part of the master's degree
in information security

Department of Computer Science

2024

Supervisor

Prof. Dr. João Resende, Faculty of Sciences of the University of Porto

UNIVERSIDADE DO PORTO

MASTERS THESIS

Syscall Security Components

Author:

Henrique LÉO

Supervisor:

João RESENDE

*A thesis submitted in fulfilment of the requirements
for the degree of MSc. Security Information*

at the

Faculdade de Ciências da Universidade do Porto
Departamento de Ciência da Computação

December 5, 2024

*“ Faith and reason are like two wings on which the human spirit rises to the contemplation of truth;
and God has placed in the human heart a desire to know the truth. ”*

Fides et Ratio, written by Saint John Paul II

Acknowledgements

Firstly, I would like to thank God for making it possible for me to serve him and for helping me through this process. After that, I would like to thank the Holy Mary for always guiding me.

I want to thank my parents, Adriana and Bruno, for always supporting me, encouraging me, and advising me to achieve all my goals. I love you with all my heart.

I want to thank my beautiful Wife, Luísa, who began this journey as my fiancée and whom I was fortunate enough to marry during this process; thank you for always being concerned for my health, for the bad nights sleeping, and for your patience so that I could make this possible.

To both my brothers, Maria Clara and João Guilherme, whom always drove me to be better.

To my beloved daughter, Teresa, may I be as dedicated to you as I was to this thesis.

UNIVERSIDADE DO PORTO

Abstract

Faculdade de Ciências da Universidade do Porto

Departamento de Ciência da Computação

MSc. Security Information

Syscall Security Components

by [Henrique LÉO](#)

This thesis examines the importance of intrusion detection systems in cyber security, primarily using system calls as the data source detection mechanism leveraging their capabilities as the white blood cells of a system [1]. The research details the techniques, types, and applications of HIDS, explicitly exploring how machine-learning models can be used to detect zero-day attacks. A HIDS architecture is proposed, using a publicly available dataset to develop and train a machine learning model to detect novel attacks. The architecture is tested in real-world scenarios, providing valuable information on the actual effectiveness of such a model and how it would be helpful in the security of a host also a novel approach on explaining the detection is proposed of anomaly models so that the analyst can better understand the Alerts. The study displays the strengths and limitations of existing datasets and machine learning methods and the challenges of applying these methodologies in the information technology ecosystem. The results highlight the importance of careful analysis when translating advances in data science into usable security tools, their limitations, and how the system call is still one of the primary methods of detecting anomalies in a system.

UNIVERSIDADE DO PORTO

Resumo

Faculdade de Ciências da Universidade do Porto

Departamento de Ciência da Computação

Mestrado em Segurança Informática

Componentes de Segurança de Chamadas de Sistema

por [Henrique LÉO](#)

Esta tese examina a importância dos sistemas de deteção de intrusões na cibersegurança, utilizando principalmente as chamadas de sistema como mecanismo de deteção de fontes de dados, tirando partido das suas capacidades como glóbulos brancos de um sistema [1]. A investigação descreve em pormenor as técnicas, os tipos e as aplicações dos HIDS, explorando explicitamente a forma como os modelos de aprendizagem automática podem ser utilizados para detetar ataques de dia zero. É proposta uma arquitetura HIDS, utilizando um conjunto de dados publicamente disponível para desenvolver e treinar um modelo de aprendizagem automática para detetar novos ataques. A arquitetura é testada em cenários reais, fornecendo informações valiosas sobre a eficácia real de um modelo deste tipo e a sua utilidade para a segurança de um anfitrião. Além disso, é proposta uma nova abordagem para explicar a deteção de modelos de anomalias, para que o analista possa compreender melhor os alertas. O estudo mostra os pontos fortes e as limitações dos conjuntos de dados e dos métodos de aprendizagem automática existentes e os desafios da aplicação destas metodologias no ecossistema das tecnologias da informação. Os resultados destacam a importância de uma análise cuidadosa ao traduzir os avanços da ciência dos dados em ferramentas de segurança utilizáveis, as suas limitações e a forma como a chamada do sistema continua a ser um dos principais métodos de deteção de anomalias num sistema.

Contents

Acknowledgements	v
Abstract	vii
Resumo	ix
Contents	xi
List of Figures	xiii
1 Introduction	1
1.1 Objectives	2
1.2 Contributions	2
1.3 Outline	3
2 Literature Review	5
2.1 Data Sources	10
2.1.1 Windows System Data Source	12
2.2 Feature Extraction Techniques	15
2.3 Datasets	16
2.4 Classifiers	19
2.5 Explainable AI Frameworks	22
2.6 Threats	23
2.7 Security Tools	24
2.7.1 Sysdig	24
2.7.2 Wazuh	24
2.8 Security Operations Center	27
3 Proposed Architecture	29
3.1 Hardware and Software requirements	30
3.2 Anomaly Detection Model	30
3.2.1 Framework	30
3.2.2 Dataset	31
3.2.3 Data Preprocess	33
3.2.4 Decision Engine	37
3.2.4.1 Training	37

3.2.4.2	Testing	39
3.3	Host Evaluation	40
3.4	SIEM Integration	42
3.4.1	Explanation Model	43
3.4.2	Wazuh Alert	44
3.5	Concerns	46
4	Evaluation	47
4.1	Preparation Phase	47
4.2	Simulation Phase	48
4.2.1	Container Evaluation	48
4.2.1.1	Container Normal Behavior	48
4.2.1.2	CVE-2021-42013	49
4.2.1.3	CVE-2022-33891	52
4.2.1.4	CVE-2022-42889	52
4.2.1.5	CVE-2023-7172	54
4.2.1.6	CVE-2023-30212	55
4.2.2	Host Evaluation	56
4.3	Results	59
5	Conclusion	61
5.1	Research Summary	61
5.2	Current Limitations	62
5.3	Future Work	63
5.4	Conclusions	63
A	Appendix A	65
B	Appendix B	67
	Bibliography	71

List of Figures

2.1	Generalized architecture of misuse-based and anomaly-based HIDSs [7]. . .	6
2.2	System Call architecture for Linux Systems [25]	11
2.3	System Call Based HIDS Architecture [26].	11
2.4	High-Level Overview of how a syscall is performed in Windows Systems [29]	13
2.5	Windows DLL Architecture [34]	14
2.6	Comparison of HIDS datasets.[53]	19
2.7	Sysdig Architecture [25]	25
2.8	Wazuh Dataflow Architecture [70]	26
3.1	Architecture proposed for Thesis.	29
3.2	Syscall Information on other Datasets.	32
3.3	Syscall traces of LID-DS 2019.	32
3.4	LID-DS is Original Layout	33
3.5	Example of output of the Explainer	45
4.1	Attack used in HTTP Apache server	50
4.2	Syscall Traces of regular usage of the Apache HTTP server	50
4.3	Syscalls traces of attack and the generation of the alert.	50
4.4	Wazuh Dashboard with the anomaly detection alert	51
4.5	Syscall trace of the Alert for the Exploitation of CVE-2021-42013.	51
4.6	Graphic with the Shap Values of the detection.	51
4.7	Exploitation of the CVE-2022-33891	52
4.8	Syscall trace of the Alert for the Exploitation of cve-2022-33891.	53
4.9	Explainer Output for the RCE of the Vulnerability cve-2022-33891	53
4.10	Syscall trace of the Alert for the Exploitation of cve-2022-42889 for RCE. . .	54
4.11	Explainer Output for the RCE of the Vulnerability cve-2022-42889	54
4.12	Syscall trace of the Alert for the Exploitation of cve-2022-42889 for SSRF. . .	54
4.13	Explainer Output for the SSRF of the Vulnerability cve-2022-42889	54
4.14	Syscall trace of the Alert for the Exploitation of cve-2023-7172.	55
4.15	Explainer Output for the SQLi of the Vulnerability cve-2023-7172.	55
4.16	First Syscall trace of the Alert for the Exploitation of cve-2023-30212.	56
4.17	First Syscall trace of the Alert for the Exploitation of cve-2023-30212	56
4.18	First explainer Output for the XSS of the Vulnerability cve-2023-30212. . . .	57
4.19	Second explainer Output for the XSS of the Vulnerability cve-2023-30212. . .	57
4.20	Alerts displayed using only signature-based detection	58
4.21	Two Nmaps Executions	58
4.22	Nmap syscall trace detected as an Anomaly by the Architecture	59

4.23 Explanation of the Nmap detection	59
--	----

Chapter 1

Introduction

Securing information technology systems is highly important in the current world, where every aspect of society's day-to-day activities, such as banking, health care, and supermarkets, rely on these systems. However, given this reliance on the systems, many people and organizations attempt to exploit them and interrupt their correct behavior to gain unlawful access, financial benefits, or control over sensitive data. These exploits can manifest in various forms, such as hacking, malware attacks, phishing, and ransomware. To prevent and mitigate such consequences, organizations create security incident plans that help respond to such occurrences, which unfortunately cannot always be avoided.

Detection is a crucial element in any security incident response plan [2]. Organizations heavily rely on detection systems to detect intrusions, so much so that there are dedicated budgets and entire teams to respond to the detections made by these systems. So, the quality and reliability of these systems are crucial to any mature security management system.

However, detecting attacks is far from straightforward. It requires a delicate balance between leniency and scrutiny. Too much leniency can allow malicious actors to operate within the network undetected, posing a threat to data integrity, availability, and confidentiality. On the other hand, excessive scrutiny can flood security teams with a high volume of detections, many of which are not malicious, leading to fatigue and a lack of response time for genuine invasion events.

The intrusion detection field has multiple segments, with its primary division being between the Network Intrusion Detection system (NIDS), which monitors network traffic for suspicious activity, and the Host Intrusion Detection System (HIDS), which analyzes individual devices for anomalies[3]. This thesis will focus on Host intrusion detection

schemes, the type used, the strategies implemented, and how they can be leveraged. Although the HIDS only analyzes a single device, it has much more data and complexity than the NIDS, which relies solely on network packets.

1.1 Objectives

The three primary objectives of this thesis are the comprehensive exposition of the history of the HIDS strategies, implementations, types, and challenges, primarily focused on the use of syscalls as the significant source of the detection capability of said systems.

The design of a HIDS architecture that leverages a machine learning model that uses a publicly available dataset of syscalls as its data source and also creates an explainable AI model where that will help the analyst in the handling of the alert of novel attacks that do not have any signatures available, highlighting how the syscalls and features extracted from the syscalls can create a baseline of the system's behavior, and thus increase the security level of the host and the capability in detecting attacks, specifically remote code executions by unauthorized sources. Also, it highlights how the architecture's effectiveness relies on how the data was collected from its dataset by demonstrating the different scenarios on only monitoring container environments or the whole system.

The architecture will be rigorously evaluated using real-world attacks and scenarios. These scenarios will include exploitation of CVEs that were not available when the dataset was created and malicious behaviors that are hard to create a signature for. This evaluation will test the dataset and the machine learning models in a real-world application, focusing on usability in the cybersecurity realm.

1.2 Contributions

The main contributions of this thesis are as follows:

- A novel approach using a modern publicly available dataset for host intrusion detection to develop a machine learning model capable of identifying previously unknown exploits and evaluating the models effectiveness in detecting code execution attacks on containers and full system environments.
- An investigation into the performance of synthetically generated datasets in real-world scenarios, which are far more complex than controlled lab environments. An

analysis of how datasets collected using containers or other virtualization tools affect model performance highlights such datasets' limitations in real-world applications.

- The application of a logical algorithm for anomaly detection reduces the false positive rate in models trained with synthetic data, thus minimizing unnecessary alerts and improving model accuracy.
- The integration of the proposed model into an open-source monitoring platform enhances the analysis of alerts and streamlines the incident response process.
- The novel application of an Explainable Artificial Intelligence (XAI) framework to assist security analysts in interpreting anomaly detection alerts generated by the HIDS. It is achieved by utilizing SHAP mean values in detection traces, providing insight into otherwise black-box detection processes.

1.3 Outline

This thesis is divided into five chapters: Chapter 1 outlines the motivations and objectives of the thesis and how it contributes to a better understanding and improvement of the current cybersecurity landscape.

Chapter 2 presents a comprehensive Literature Review regarding the HIDS landscape. It covers the major papers that have contributed to improving techniques and data sources, as well as the machine learning techniques and frameworks that have influenced the data science and security fields. Additionally, it provides an overview of modern open-source tools used to detect modern threats.

Chapter 3 presents the design of an architecture that was constructed based on a publicly available dataset and framework, the techniques used to evaluate the syscalls of the systems, how was created the model used, and how the integration between security tools and the model done to ensure that the SOC analyst is capable of responding to the detections.

Chapter 4 evaluates the model designed in Chapter 3, testing it against vulnerabilities that the model was not trained on. This evaluation also assesses how the information in the generated alert assists the incident responder in evaluating the root cause of the detection. The practical implications of this evaluation are intended to assess whether or

not a publicly available dataset can have an actual application in the real world, not only in the data science field.

Finally, chapter [5](#) discusses the research results and the objectives that were met, discussing future work that might improve the field in HIDS research.

Chapter 2

Literature Review

Intrusion Detection Systems (IDS) represent a critical component in the multi-layered security architectures employed by modern infrastructure. As the last line of defense, IDS aims to detect adversaries who have bypassed other security controls and are operating within the network or system. Given the rapidly evolving landscape of Information and Communication Technology (ICT) systems, accurately distinguishing between benign and malicious activities remains one of the primary challenges in IDS research, as these systems must operate in environments characterized by high complexity and dynamic configurations [4].

IDS can broadly be divided into Network Intrusion Detection Systems (NIDS) and Host-based Intrusion Detection Systems (HIDS), each with distinct capabilities and limitations. NIDS primarily monitors network traffic, analyzing packets and flows at strategic points. At the same time, HIDS operates on individual endpoints, drawing on a wider array of data sources such as system logs, registry keys, and system calls [5]. Despite these functional differences, the core challenge of IDS accurately detecting sophisticated attacks while minimizing false positives remains consistent across both types.

Traditionally, IDS have employed signature-based (or misuse-based) detection techniques, which rely on predefined patterns of known attacks. While effective at detecting previously observed threats, signature-based methods are inherently limited in detecting zero-day attacks or novel threats that do not match known signatures. In a world where advanced persistent threat (APT) groups routinely employ obfuscation techniques to evade detection, the limitations of signature-based systems are increasingly exposed [6]. Zero-day exploits and polymorphic malware can bypass such systems, leaving critical gaps in network defenses.

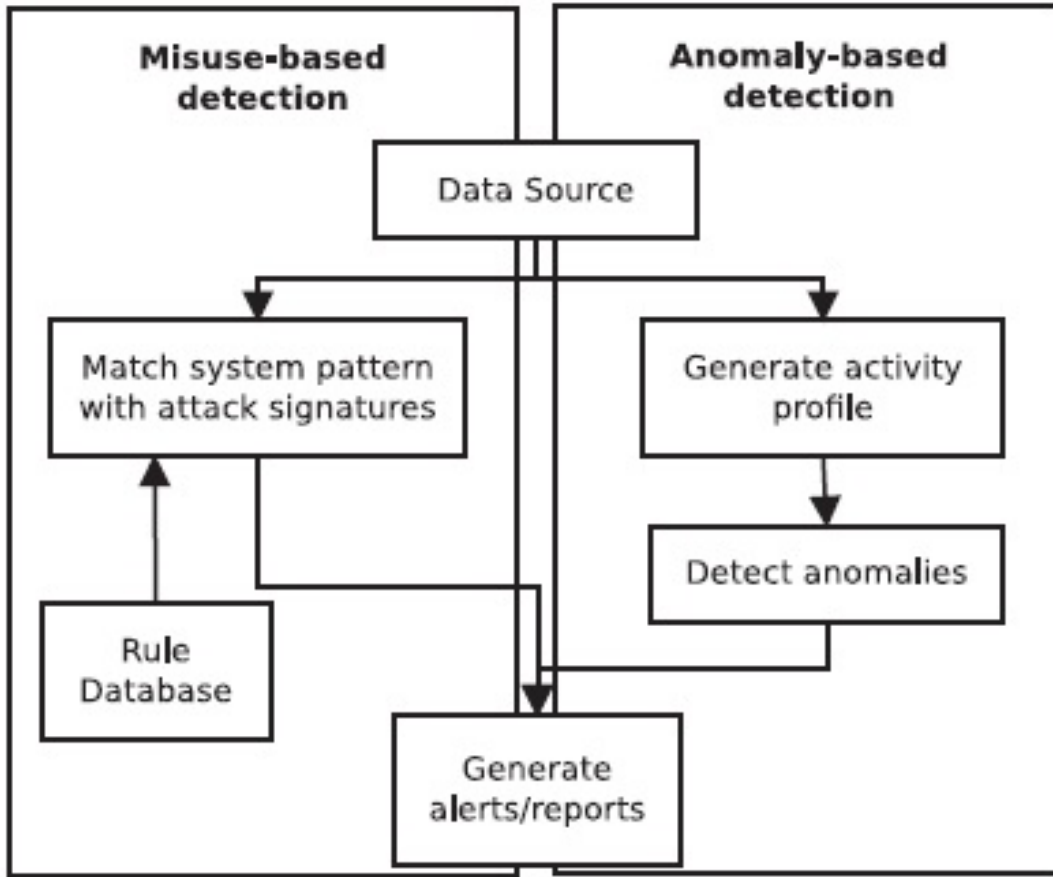


FIGURE 2.1: Generalized architecture of misuse-based and anomaly-based HIDSs [7].

As a result, research has shifted towards anomaly-based detection systems, which aim to detect deviations from established behavioral baselines. Anomaly-based IDS can theoretically detect previously unknown attacks by learning the normal behavior of systems and networks. A summary of the differences between the two approaches can be seen in Figure 2.1. However, the practical application of these systems is hindered by high false alarm rates (FAR) due to the variability in legitimate behavior. Striking a balance between detection rate (DR) and FAR is a persistent issue, as overly strict thresholds can generate excessive false positives. At the same time, lenient models risk missing genuine threats [6, 8]. This balance is challenging to achieve in highly dynamic environments where the baseline of "normal" behavior is constantly shifting.

$$DR = \frac{\text{number of detected attacks}}{\text{number of attacks present}} \times 100\% \quad (2.1)$$

$$FAR = \frac{\text{number of false alerts}}{\text{number of attacks present}} \times 100\% \quad (2.2)$$

While metrics like the ROC-AUC curve are widely used to evaluate IDS performance, these metrics can be misleading when applied to imbalanced datasets, which are common in IDS research. Some researchers have proposed alternatives, such as the Matthews Correlation Coefficient (MCC), which is more robust in such scenarios. However, the adoption of MCC in IDS literature remains limited, signaling a need for broader evaluation frameworks to more accurately capture real-world IDS deployment challenges [9, 10].

An often overlooked aspect of IDS research is the issue of time and space complexity. Many state-of-the-art models, particularly those employing machine learning techniques, require significant computational resources for training and execution. This poses a considerable obstacle to real-time IDS deployment in environments where latency and resource constraints are critical concerns. Despite this, there is a notable gap in the literature regarding evaluating IDS solutions in terms of their impact on system performance, such as latency and CPU utilization [11, 12].

Hybrid models, which combine signature-based and anomaly-based techniques, have emerged as a promising solution to the limitations of each approach. By leveraging the strengths of both methods, hybrid IDS aim to detect known and unknown attacks while reducing the overall false positive rate. These systems also integrate additional data points, such as CPU and memory usage, file integrity checks, and statistical analysis of system behavior, to enhance detection capabilities [13, 14].

Modern HIDS typically follow a framework where raw audit data such as system calls in Linux-based systems are collected and processed to extract meaningful features. These features are then fed into decision engines, often based on machine learning models, to classify behaviors as benign or malicious [8]. One of the earliest models, STIDE, exemplifies this approach by using a sliding window technique to compare system call sequences against a database of known patterns, flagging deviations as potential threats [1, 15]. However, while such methods have proven effective in controlled environments, their applicability to real-world systems remains constrained by the challenges of maintaining accurate baselines and minimizing false positives.

Another critical challenge in HIDS is the lack of transparency in machine learning-based decision engines, often called "black box" algorithms. While these models can improve detection rates, they provide little insight into the reasoning behind specific alerts,

making it difficult for analysts to assess and respond to potential threats. This opacity, combined with the high volume of alerts generated by anomaly-based systems, contributes to the growing problem of alert fatigue, where security personnel become overwhelmed by false positives and fail to respond to genuine incidents promptly [16, 17].

Table 2.1 presents a comprehensive overview of notable Intrusion Detection System studies, highlighting the methodologies used, the data sources, and the unique contributions of each work. The studies cover various approaches, such as sequence analysis, data mining, machine learning, and deep learning, applied to both host-based and network-based intrusion detection systems. System calls and network packets are the most common data sources, with some studies using alternative sources like system logs. The contributions range from foundational techniques like anomaly detection in Unix processes to modern deep learning-based architectures for detecting network intrusions, emphasizing the evolution of IDS technologies over time.

In conclusion, while IDS technology has advanced significantly in recent years, several critical challenges remain. The ability to accurately detect emerging threats, balance detection performance with system resource constraints, and provide actionable insights to security analysts are crucial areas of ongoing research. The development of hybrid systems, the application of advanced machine learning techniques, and the exploration of new evaluation metrics are all promising directions for the future of IDS research. Nevertheless, the gap between theoretical models and practical deployment in complex, real-world environments remains a significant hurdle for researchers and practitioners alike.

Author	Title	Data Sources	Contributions
Forrest et al. [1]	A Sense of Self for Unix Processes	System Calls	Introduced the idea of using short sequences of system calls to detect anomalies in Unix processes, establishing the foundation for host-based IDS.
Stolfo et al. [18]	Cost-based modeling for fraud and intrusion detection: Results from the JAM project	Network Packets	Applied data mining for IDS, using association rules and frequent episodes to detect network intrusions.
Lippmann et al. [19]	Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation	Network Packets and System Logs	The paper established a standardized evaluation framework for IDS, using the 1998 DARPA dataset to benchmark network and host-based IDS performance, highlighting strengths and weaknesses in detection capabilities.
Axelsson [20]	The Base-Rate Fallacy and the Difficulty of Intrusion Detection	System Calls	Presented a comprehensive taxonomy of IDS and discussed challenges related to false positives in anomaly-based detection.
Sommer and Paxson [21]	Outside the Closed World: On Using Machine Learning for Network Intrusion Detection	Network Packets	Discussed the practical challenges in applying ML to real-world network-based IDS, with emphasis on scalability and false positives.
Creech [14]	Developing a High-Accuracy Cross Platform Host-Based Intrusion Detection System Capable of Reliably Detecting Zero-Day Attacks	System Calls	Demonstrated the effectiveness of cross-platform host-based IDS using hidden Markov models (HMM) and LSTM for detecting zero-day attacks.
Roesch [22]	Snort Lightweight Intrusion Detection for Networks	Network Packets	Introduced Snort, a widely used open-source IDS for signature-based detection of known attacks, still prevalent today.
Ravi et al. [23]	Applying convolutional neural network for network intrusion detection	Network Packets	Comprehensive deep learning-based IDS comparing CNN, RNN architectures applied to network traffic data.

TABLE 2.1: Summary of IDS studies, methodologies, data sources, and contributions

2.1 Data Sources

Data sources for HIDS may vary depending on the Operating System in question. However, researchers agree that looking for the rawest and most unfiltered data is the best way to create a behavioral baseline for a system, given that no interpretation or abstraction is added. Hence, the behavior is not dependent on a third party's interpretation.

For Unix-based systems, it is universally recognized that system call trace is the most significant source for leveraging the raw information of the system to detect anomalies. This understanding, derived from the work of Forrest, is the most straightforward interface between a program and the kernel. Any process that aims to perform a meaningful operation must first reach the kernel. Initially, Forrest used the Sequence Time-Delay Embedding (STIDE) methodology [1], which only observed syscall frequency for a specific program. This approach, while not scalable due to the large number of data needed to be effective, was prone to attacks such as the mimicry attack [24], where the attacker observes the systems normal patterns and crafts the attack to mimic legitimate operations, allowing them to bypass detection and carry out malicious activities undetected. This makes it challenging for traditional security mechanisms to identify the attack. However, Forrest's work [1] laid the foundation for the field of HIDS. In the following years, many researchers used syscall as the foundational piece in applying different methods to extract algorithms that would detect anomalous behavior. Figure 2.2 illustrates how the system call interacts with the Unix kernel. A system call (syscall) is a mechanism that allows a program in User Mode to request services or resources from the operating system, such as file handling, memory management, or process control. It is crucial because it provides controlled and secure access to critical hardware and system resources. It ensures stability and security in multi-tasking environments by isolating user processes from directly accessing the kernel.

Figure 2.3 shows the architecture of a HIDS that has as its primary source of data for analysis the syscall traces produced by the programs that run on the host OS, where when transitioning from user to kernel mode, it creates the raw data needed to detect anomalous behavior.

Another important aspect of an HIDS based on syscall traces is how the model will be trained if a single program trains the model or the model is trained by the whole system. In the first approach, each program inside the host will have a dedicated trained model

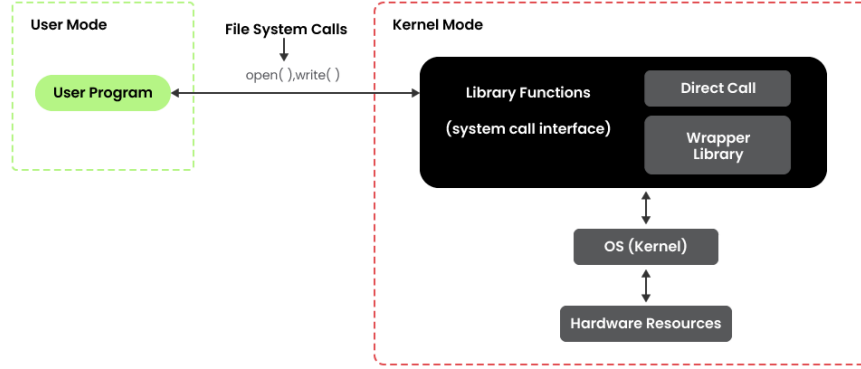


FIGURE 2.2: System Call architecture for Linux Systems [25]

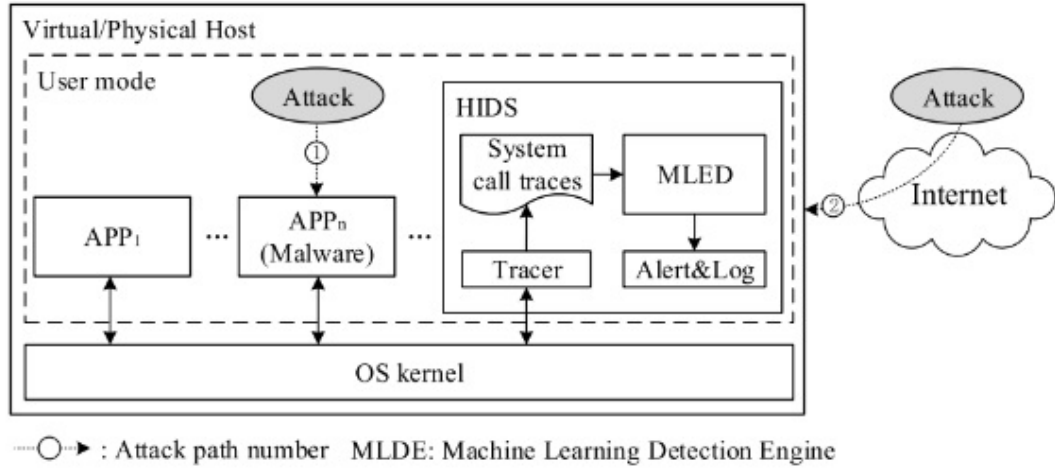


FIGURE 2.3: System Call Based HIDS Architecture [26].

where the model will have a baseline of normal behavior for each program, or in the second. The HIDS will train itself on the holistic of the system and create a model based on the system's behavior. Logically, given that modern systems have hundreds or even thousands of programs, training a model for each program is unfeasible and would create many performance issues, so most modern HIDS is focused on a holistic view of the system [8], is important to note the Forrest, for example, trained its model for only a single program. Some datasets are explicitly created for a single program, such as a dataset specifically for the Firefox program. [27]. So, choosing which approach to use depends on the need to detect the intrusion, given that if the model is trained for a single program, the accuracy for that program will be much higher than if a HIDS is trained for the entire system.

The syscall method has some issues, especially in modern virtualized environments with containers. Programs, especially multi-threaded ones, make many syscalls every second. This makes it difficult to evaluate a program using a sequence of syscalls due to parallel execution. For example, using the STIDE method is now outdated. Another problem is that collecting this many syscalls in real time for each program creates a lot of overhead, which can drain the host's resources. This can be more harmful than helpful in systems requiring high performance or time-sensitive systems [12, 28]. Deciding the trace length can undermine the DE's capability to have good accuracy, given that the HIDS might not respond promptly if a trace is too long [14].

2.1.1 Windows System Data Source

While the syscall sequence analysis is considered the best source of raw information, sometimes this data is unavailable due to OS limitations. So, other data sources are used to create the anomaly models for the HIDS. These data sources are log files, registry access information, and DLL Calls. For example, the Windows OS, the most popular personal computer operating system, does not use the same architecture as UNIX. In contrast, the program calls the kernel directly via Syscall in a Linux environment. Windows systems rely on a Dynamic Linked Library (DLL) architecture, so the Syscall is not accessible nor represents the system activity in the same way as UNIX systems [14].

The use of log files in the implementation of HIDS is not a novel idea [30]. Even though logging is something that every operating system does in some capacity, the log itself has already interpreted information that was parsed and manipulated by a logging program and may cause a loss of information, given that the primary objective of the logging program is not creating the best information for the IDS designer, and more so the tinkering with the logs of a system is part of the techniques of adversaries [31]. Using this data, the model itself is susceptible to upstream obfuscation by the loggers themselves [11].

Registry-based HIDS [32] is also an alternative, given that many Windows programs frequently interact with registry keys, so examining registry access might indicate malicious behavior. However, some flaws in this approach are that it is equal to the log analysis; registry manipulation is one of the techniques to evade detection [33], and also critical execution might not need access to the registry, creating a blindspot for HIDS that only rely on the registry changes.

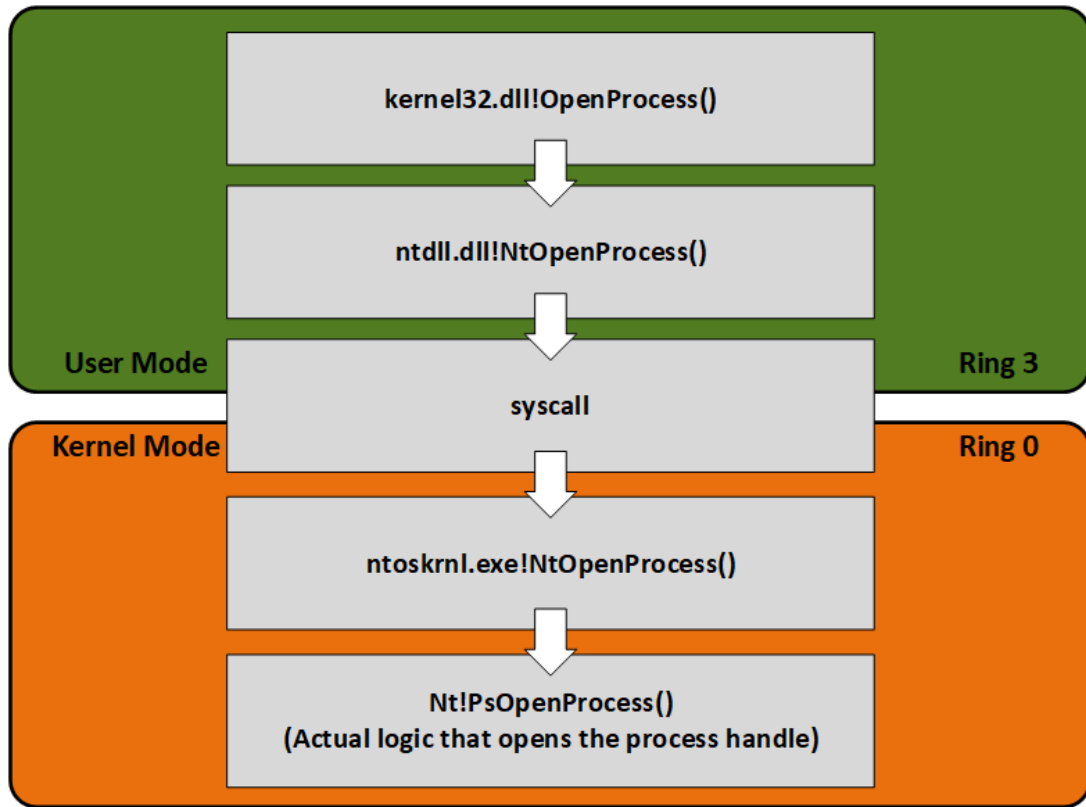
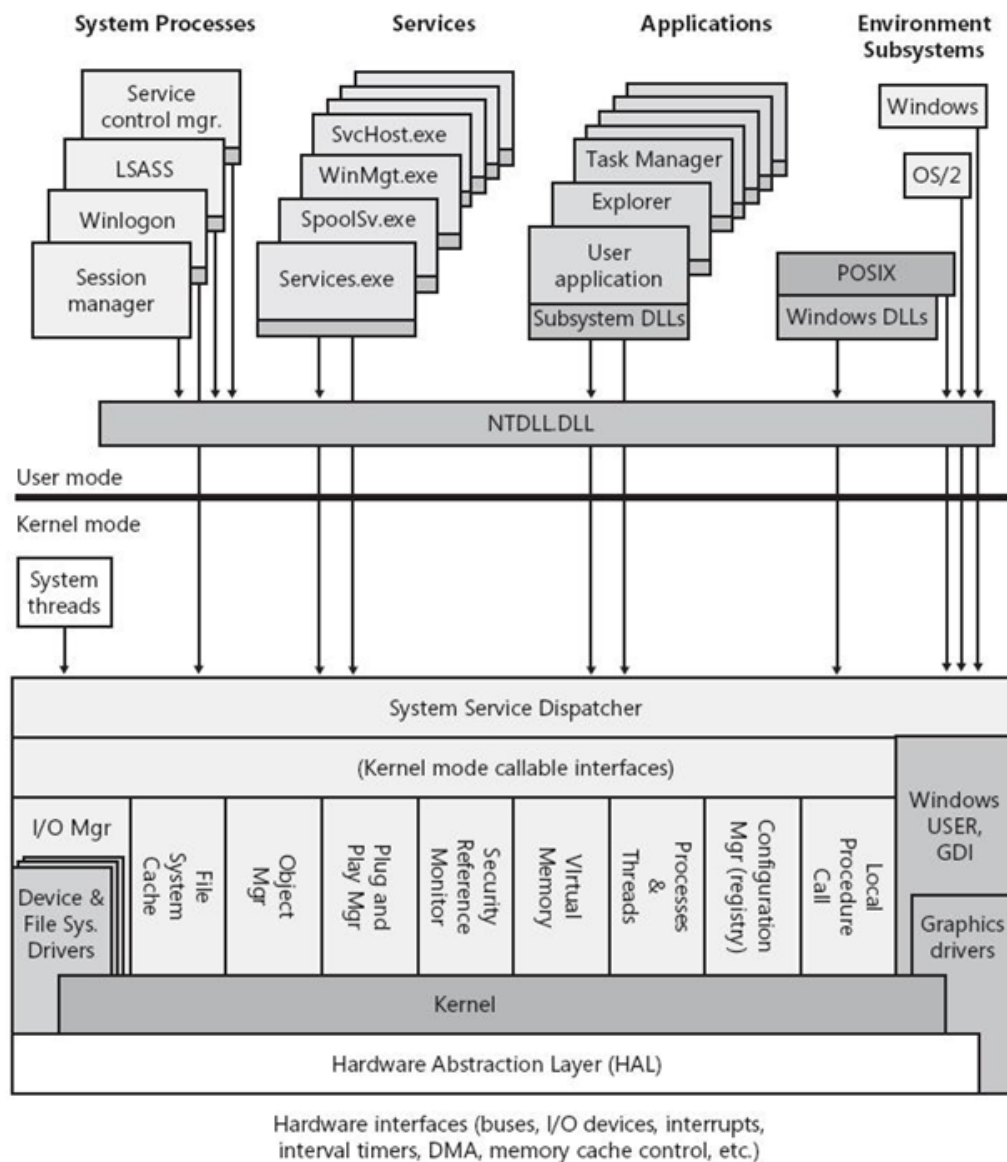


FIGURE 2.4: High-Level Overview of how a syscall is performed in Windows Systems [29]

DLLs Access request was a novel introduction in [14], where the researcher elected 9 DLLs (ntdll.dll, kernel32.dll, user32.dll, comctl32.dll, ws2-32.dll, mswsock.dll, msvcrt.dll, msvcpp.dll, and ntoskrnl.dll) to monitor, and in doing so, when programs call a DLL, the DLL will use a syscall or other system architecture aspect the necessary footprints of a program inside the Windows OS, so monitoring DLLs requests would be the equivalent of monitoring syscall traces on UNIX systems, however with a higher level of abstraction, this proved to be very efficient, and many Endpoint Protection Platforms adhered to this type of architecture. However, it still has some shortcomings compared to the more well-established syscall traces; the one obvious one is that if the program does not use one of the DLLs being monitored, the HIDS is ineffective.

Also, many modern attacks subvert this technique by bypassing the DLLs completely and calling system operations directly using techniques such as Direct or Indirect syscalls and many other low-level attacks that avoid the security products. [29] For these attacks, a kernel driver can be implemented to closely monitor the kernel's behavior. However, due to the difficulties in creating kernel drivers in Windows and also the closed-source nature of the Operating system, there is no research nor public data set with these telemetries



Reprinted, by permission, from *Inside Microsoft Windows 2000, 3rd Edition* (ISBN 0-7356-1021-5). © 2000 by David A Solomon and Mark E. Russinovich. All rights reserved.

FIGURE 2.5: Windows DLL Architecture [34]

that could explore the use of the syscalls in anomaly detection [35, 36].

2.2 Feature Extraction Techniques

In the seminal work of Forrest [1], the only feature extracted from the syscall trace was the sequence of IDs of the syscall and their relative position inside the trace of a single program. It is important to point out that Forrest conceptualized that the features extracted from the syscall needed to be simple, and too much complexity would create an unnecessary abstraction, given that feature extracted from the syscall would represent the equivalent of white cells of a human body to the computer, so simplicity is a critical aspect to the design of the features of a HIDS.

The following research explored the use of other types of features that a syscall might present to the IDS architect; other studies, such as [37–40], also used the sequence of syscall IDs and their frequency based as the primary feature in building their detection models.

Early models also used the frequency of unique short sequences of system calls as features, where a vector was constructed with labels of frequency of each system call, reducing the amount of data that a Decision Engine would consume, however as seen in [41, 42], both these approaches, sequence of system call IDs and frequency of system call, have their shortcomings, since these approaches are not accurate in complex attacks with little and dispersed footprints.

Some HIDS researches [43, 44], also explored the use of the parameters of syscalls to establish the baseline and identify any deviation; these parameters include the execution path of the syscall, variables of system call routines, the input/output of a syscall. Even though giving more context of a syscall to the model yields positive results in determining an intrusion, the feasibility of this approach in a real-world scenario is questionable due to the significant overhead in collecting this large pool of information of the system and rules that extract this kind of information must be manually updated if any major updates occur on the system.

Another approach is to limit the Syscalls sequences using domain knowledge. The domain experts choose syscall sequences that are more related to attacks and that are privileged and are the ones being used in known attacks, or might create other abstractions with the use of kernel to reduce the number of syscalls; this approach dramatically reduces the size of data, the processing time of training by the DE. However, given the

ever-changing nature of attacks and systems, the syscall that was once important in attacks might not be in the future, thus limiting the visibility of the HIDS in a real-world application [45–47].

Semantic extraction was a novel approach introduced in [11] where a dictionary is constructed using the semantic-feature-construction technique in syscall; even though this approach showed superior performance in accuracy, a significant drawback is that to build a reliable dictionary based on the semantics of the syscall it may take weeks and does not help create a reliable HIDS in a scenario where it has to face gigantic amounts of syscall data and many complex attacks.

Another approach in recent years is the use of Word embedding, which is a method for representing sequences that captures semantic relationships and contextual information from system calls using smaller vectors. This approach offers better generalization capabilities compared to n-gram-based methods. Security researchers use well-known libraries to implement this feature extraction technique by effectively capturing semantic features, such as Word2Vec, GloVe, FastText [48, 49]. One of the drawbacks to this technique is point out at [50], where it can introduce duplicates in the data sets of HIDS and thus create an overly optimistic view of the models developed by researchers since the duplicates might be in the training and the testing datasets.

2.3 Datasets

An IDS dataset is used to train diverse models to better approximate the decision engines to the intended behavior scenarios, which contain normal behavior and attacks. Datasets can differ in what kind of data source they contain and the metadata associated with them, such as syscalls and logs, whether they are public or private, or whether the dataset's data is simulated or real.

Most datasets used in the HIDS realm have the syscall traces as the primary source. Given its robust capabilities that were expanded in section 2.1, one of the main difficulties in creating a dataset is that if the dataset is in a controlled environment, the normal data and attack data are not representative of what a real-world behavior might generate. In doing so, the decision engines have some bias. They might not correspond to a real-world scenario, creating concerns about the reliability of these simulated datasets in a cybersecurity operation. Also, another problem is that the datasets are very imbalanced, tending to

have a lot of normal traces and not so many attack traces. Most publicly available datasets are outdated and do not correspond to the modern threat scenario. [10, 12].

1. *KDD* It is the first known effort to generate a public dataset to develop an IDS system released by the Defense Advanced Research Projects Agency. (DARPA) [19] The dataset has information that goes from the OS logs to network packet captures and was used to train HIDS and NIDS. Given that the dataset was generated in 1998, it is considered outdated, given the major differences in the usage of computer systems in the present day compared to the time. This dataset's data also comes from simulated behaviors; both normal and attack traces are from a controlled environment.
2. *UNM* another Dataset that is already considered legacy is the UNM dataset, created by researchers at the University of New Mexico; the focus of this dataset was capturing the Syscall of different programs on a Solaris OS, thus providing a dataset to create a per process HIDS. This dataset's data also comes from simulated behaviors; both normal and attack traces are from a controlled environment.
3. *ADFA-LD* The Australian Defence Force Academy released this data set [51] and consists of traces collected from a vulnerable Linux instance where the system was attacked with, at the time, modern penetration testing tools by security experts, developed at 2013 the attacks and techniques might be already outdated, given that it's been almost ten years since its creation. However, as per [10], this dataset is still the most used by researchers to train HIDS.
4. *ADFA-WD* This dataset was also released by the Australian Defence Force Academy; however, it focused on collecting modern intrusion attempts on Windows system; it used the novel feature DLL calls [14] as the data for the dataset, similar to the ADFA-LD it does not represent a modern attack anymore.
5. *NGIS-DS* Was also released by the Australian Defence Force Academy [8] in 2017; the main objective of the dataset was to create a highly realistic dataset so it could replicate real-world scenarios with much more information than the ADFA-LD, however, due to some inaccuracy in the timestamps of the dataset it makes impossible to reconstruct accurate syscall traces, and thus its hard to train models that are dependent on time context.

6. *AWSCTD* The Attack-Caused Windows System Calls Traces Dataset is a Windows-based dataset that was created by running real-world malware collected by Virus Share and VirusTotal on the Windows 7 OS and collecting the syscalls. [52]
7. *LID-DS* The Leipzig Intrusion Detection - Data Set is one of the most modern and complete publicly available datasets, containing real-world attacks and in a modern operating system (Ubuntu 18.04). It was generated using a framework developed by the researchers and made publicly available. The syscalls in the dataset are also more detailed, containing information such as parameter, return data, user ID, process/thread ID, file system handle, timestamp, and I/O buffer, making a very rich dataset compared to ADFA-LD that only had the syscalls IDs, it has two versions the first of 2019 that contains traces of normal behavior and attack behavior for ten vulnerabilities and 2021 that expanded the dataset and included five more scenarios and increased the number of captured syscalls 2.2. [53, 54]

Vulnerability Scenarios	Number of Syscalls in Millions in LID-DS 2019	Number of Syscalls in Millions in LID-DS 2021
CVE-2012-2122	5.7	20.7
CVE-2014-0160	4.0	1.9
CVE-2017-7529	1.8	1.3
CVE-2017-12635_6	N/A	1311.7
CVE-2018-3760	19.2	115.1
CVE-2019-5418	18.0	400.9
CVE-2020-9484	N/A	223.6
CVE-2020-13942	N/A	849.1
CVE-2020-23839	N/A	33.9
Bruteforce_CWE-307	5.7	9.5
EPS_CWE-434	126.2	296.3
Juice-Shop	N/A	484.9
PHP_CWE-434	22.2	97.5
SQL-injection	23.6	96.2
ZipSlip	252.1	111.1

TABLE 2.2: Number of Syscalls in LID-DS 2019 and LID-DS 2021 for Various Vulnerability Scenarios

Figure 2.6 illustrates the difference in the data quality and availability in each of the Linux Datasets, also not considering that their data is not applicable in modern environments for the older datasets.

	LID-DS	NGIDDS	ADFA-LD	UNM	KDD99
Process IDs	X	X		X	X
Arguments	X				X
Return values	X				X
Timestamps	X	(X)			X
Not outdated	X	X	X		
Data buffers	X				
Amount of data	X	X			X

FIGURE 2.6: Comparison of HIDS datasets.[53]

2.4 Classifiers

A classifier in machine learning is an algorithm that categorizes data points into different classes. Classifiers can be binary, multiclass, or multilabel. A binary classifier is used when there are only two possible classes, for example, in intrusion detection, if the trace is malicious or a normal trace. Multiclass classifiers are when there are more than two classes, and multilabel is when a data point can be in more than one class[16].

The three approaches to train the classifiers are supervised, semi-supervised, and unsupervised. Supervised training is when the model is provided with the data labels in the training data so the model knows what is considered normal and what is considered abnormal. In unsupervised training, the data has no labels, and the model has to identify patterns inside the data. Semi-supervised training is when there are two types of data, labeled and not labeled, with more commonly the unlabeled data being the most prevalent[55].

For the Decision engine of the HIDS, different classifier models were used by researchers to try to detect anomalies in the features constructed in the datasets; at the beginning of the development, used more simple algorithms such as Sequence Time-Delay Embedding (STIDE) [1], Support Vector Machines (SVM) [56], or Hidden Markov Models (HMM) [57, 58]. Other statistical approaches; however, with the advent of better hardware, there was a shift to Deep Learning methods to try and create better models for detecting anomalies, as can be seen in more recent studies in the HIDS field [10, 59].

STIDE - Sequence Time-delay Embedding It was used as the first decision engine for anomaly detection in [1]; it monitors the frequency of syscall traces using time embedding and evaluates any deviation on the expected pattern.

A Support Vector Machine (SVM) is a straightforward supervised machine learning technique for classification and regression tasks. In practice, SVM plots each data point in an N-dimensional space, where N represents the number of features or attributes in the dataset. The goal is to find the optimal hyperplane that separates the data points. Thus, it is evident that SVM is fundamentally designed for binary classification tasks; it is used in intrusion Detection primarily associated with short sequence syscalls, although it has difficulties in implementation due to the computational requirements with large datasets and fails to detect anomalies in more complex attack patterns[10, 56, 60].

Hidden Markov Models (HMMs) are stochastic models suitable for sequential data, such as system call sequences, used in intrusion detection. An HMM consists of a hidden Markov chain with a finite number of states, each associated with a set of observation probability distributions. Starting from an initial state, the model transitions between states according to transition probabilities and emits an observation symbol based on the output probability distribution of the current state. In intrusion detection, HMMs are trained with different numbers of states and evaluated on validation sets to determine the optimal state number and decision threshold. For testing, sequences of system calls are evaluated, with anomalies detected if the sequence probability is below a threshold. Longer sequences provide better anomaly detection performance. However, HMMs are computationally intensive, requiring significant processing time, and they often produce varied false positive rates across different programs. To mitigate processing time, duplicate sequences are removed, significantly reducing the data size. Despite these efforts, HMMs remain time-consuming, making them less feasible for real-world applications [57, 58].

Ensemble methods in HIDS enhance detection capabilities by integrating multiple machine learning classifiers to analyze system call data. The predictions from these classifiers are then combined using techniques like majority voting or weighted averaging to decide whether the system activity is normal or malicious. By leveraging the strengths of various models and the rich information in syscall sequences, ensemble methods significantly improve detection accuracy and reduce false positives, leading to a more robust and reliable intrusion detection system. A drawback of ensemble methods in HIDS is the increased computational complexity and resource requirements, as they involve training and maintaining multiple models. This can lead to slower detection times, higher operational costs, and more difficult implementation and maintenance [61].

Long short-term memory (LSTM) is a specialized version of recurrent neural network (RNN) architecture used in deep learning, designed to handle long-term dependencies by maintaining feedback connections. Unlike traditional RNNs, LSTMs can process entire data sequences, making them suitable for speech recognition, pattern recognition in unsegmented data, and anomaly detection in network traffic, including intrusion detection systems (IDS). A typical LSTM unit comprises four main components: the cell, input gate, output gate, and forget gate. The cell retains values over arbitrary time intervals, while the gates control the flow of information in and out of the cell. This architecture effectively allows LSTMs to classify, process, and predict time series data, addressing the vanishing gradient problem that hinders traditional RNNs. Their ability to handle varying gaps between significant events makes LSTMs advantageous over RNNs, hidden Markov models, and other sequence learning methods in numerous applications. In intrusion detection, LSTMs are particularly useful for detecting anomalies in network traffic by leveraging their capacity to model temporal dependencies and sequence patterns[62].

The process of creating these machine-learning decision engines can be divided into the following steps [16]:

1. The training phase is where the model learns from the data. The model is fed input data sequences and the correct outputs (targets). As the data passes through the network, the model makes predictions and compares them to the actual targets. The difference (error) is calculated and used to adjust the model's internal weights and biases through backpropagation. This adjustment helps the model improve its predictions over multiple iterations, called epochs.
2. The validation phase checks how well the model generalizes to new, unseen data. A separate set of data, not used in training, is used for this purpose. During training, the model's performance on this validation set is monitored. This helps in tuning the model's hyperparameters, like the learning rate or the number of layers, to ensure that the model is not just memorizing the training data but is also capable of performing well on new data.
3. The testing phase evaluates the final performance of the trained model. Another separate dataset, called the test set, is used, which the model has never seen before. The model's predictions on the test set are compared to the actual targets to assess

its accuracy and effectiveness. This phase provides an unbiased evaluation of the model's performance and readiness for real-world application.

2.5 Explainable AI Frameworks

Explainable AI (XAI) is a set of methods and techniques in artificial intelligence (AI) that aim to make the outputs and operations of machine learning models understandable to humans. Its key objective is to ensure that AI decisions are transparent, interpretable, and trustworthy. By providing insights into how AI models make decisions, XAI not only helps users understand the reasoning behind predictions and detect biases but also significantly enhances model reliability. This is particularly crucial in sectors such as healthcare, finance, and legal systems, where a clear understanding of AI's decision-making process is essential for ethical and effective deployment [17, 63].

1. *SHAP (SHapley Additive exPlanations)* SHAP is a framework to explain the output of machine learning models. It is based on Shapley values from cooperative game theory, which assigns a value to each player (or feature) representing their contribution to the overall outcome. SHAP values provide a measure of feature importance that is both consistent and interpretable. They show how each feature influences the model's prediction by quantifying the contribution of each feature to the final prediction, ensuring that the sum of the contributions equals the difference between the actual prediction and the mean prediction. This method allows for a consistent explanation across different models and datasets [64].
2. *LIME (Local Interpretable Model-agnostic Explanations)* LIME is an approach to interpreting the predictions of any machine learning classifier by approximating it locally with an interpretable model. It focuses on understanding individual predictions rather than the entire model. LIME perturbs the input data around the instance to be explained and observes the changes in the output. It then fits a simple, interpretable model (such as linear regression) to these perturbed samples, approximating the local decision boundary of the complex model. This provides insights into which features are most influential in the local decision-making process, making it easier for users to understand and trust individual predictions [65].

Very few studies go in-depth into the use of XAI frameworks in cybersecurity, especially in alerts generated by anomaly detection solutions, given that the main objective in those

alerts is that analysts can make a decision based on the detection. None of the solutions give reasoning or an explanation as to why the detection occurs. The analyst, most of the time, only has the alert and makes assumptions of why the alert was generated, possibly discarding it, and does not take the appropriate actions for the detection, and in doing so, the anomaly detection loses its value to the Security of the infrastructure[17, 63].

2.6 Threats

Cybersecurity constantly changes, and new challenges always arrive, with attackers finding new ways to exploit vulnerabilities and bypass defenses. Public frameworks like Metasploit have democratized access to sophisticated attack tools, enabling even relatively inexperienced persons to carry out complex exploits. Metasploit, an open-source project, provides a robust suite of tools, including an extensive library of known exploits, payloads, and auxiliary modules [14, 66].

Attackers often use obfuscation techniques to evade detection by security systems. Obfuscation disguises the signatures of an attack, making it difficult for antivirus programs and intrusion detection systems to leverage signatures [12, 26] .

In addition to these challenges, advanced persistent threats (APTs) represent a significant concern. APTs involve prolonged and targeted cyber intrusions to steal sensitive information or disrupt operations. These attacks are often carried out by well-funded and highly skilled adversaries, such as nation-state actors or organized crime groups [67].

The complexity of modern computing environments exacerbates these challenges. Today's systems have numerous interconnected components, each with its potential vulnerabilities. This intricate web of dependencies complicates efforts to maintain comprehensive visibility and control over networks. Sophisticated attacks can exploit these complexities, slipping through the cracks of even the most robust security frameworks.

Another concern is supply chain attacks, where adversaries target vulnerabilities in the software supply chain to compromise widely used applications or systems. Attackers can distribute malicious code to many unsuspecting users by infiltrating trusted software vendors or updating mechanisms. The SolarWinds attack is a notable example [68], where sophisticated threat actors compromised the software update process of a widely used IT management tool, impacting numerous organizations worldwide. Also, another more recent backdoor incident was the vulnerability CVE-2024-3094 [69], where malware was covertly inserted into the widely used XZ Utils compression library, highlighting the trust

issues with significant software dependencies. This breach allowed attackers unauthorized access to systems, demonstrating the risks of relying on complex third-party components. Ensuring the integrity of extensive codebases is challenging, making it difficult for organizations to maintain trust in their software supply chains and their open-source or third-party components.

These issues highlight the ever-increasing need to rely on trust and the signature of known attacks and have mechanisms that can detect possible malicious behavior that might bypass conventional security and trust boundaries. Thus, anomaly detection HIDS is a great tool that can help prevent these attacks from being successful or avoid significant impacts with early detection.

2.7 Security Tools

This section will provide a brief overview of some of the open-source tools for HIDS that are currently available. These tools are also widely used in the following sections of this paper.

2.7.1 Sysdig

Sysdig is an open-source system monitoring tool that leverages the Extended Berkeley packet filter in the Linux kernel that allows the safe and reliable instrumentation of the kernel and gives the Sysdig tool the capability to monitor the systems calls to the kernel as illustrated in [2.7](#). The eBPF module is widely used in kernel-level applications given its reliability and low overhead, even sparking the creation of similar tooling in the Windows kernel; this tooling might help in future research in HIDS for Windows. Sysdig also offers the capability to monitor specific containers in the system and filter unwanted syscall information; thus, it becomes an excellent tool for capturing syscalls to monitor and feed to HIDS.

2.7.2 Wazuh

Wazuh is an open-source security monitoring and log management platform that helps organizations detect threats, monitor the integrity of their systems, and comply with security regulations. It integrates various security capabilities into a unified solution, making

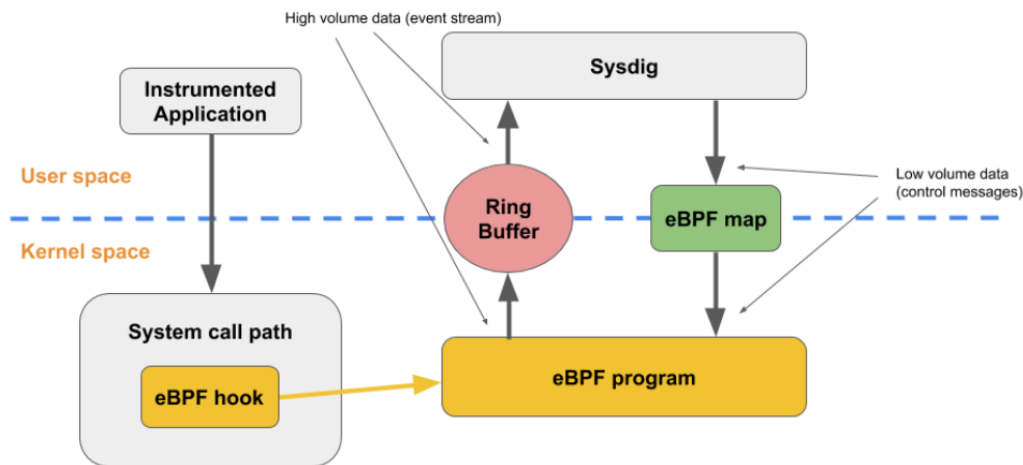


FIGURE 2.7: Sysdig Architecture [25]

it a powerful tool for a Security Operations Center (SOC). Wazuh is built with the following major components [70]:

1. *HIDS*: Wazuh uses OSSEC as its core HIDS component. OSSEC collects and analyzes system logs, monitors file integrity, detects rootkits, and triggers alerts on suspicious activities. OSSEC performs real-time log analysis, monitoring logs from multiple sources such as applications, databases, and network devices. It uses predefined and custom rules to detect and respond to potential security threats. OSSEC can assess systems' configurations against security policies and best practices, ensuring compliance and reducing vulnerabilities [71]. However, in the free tier of the product, there is no Machine Learning component for anomaly detection.
2. *ELK Stack* Elasticsearch: This component stores, searches, and analyzes large volumes of data. Wazuh sends collected logs and alerts to Elasticsearch, where they can be indexed and queried efficiently. Logstash: It processes and ingests log data from various sources into Elasticsearch. It acts as a data pipeline that formats and filters the logs collected by Wazuh before they are stored. Kibana: It provides a web interface for visualizing data stored in Elasticsearch. It allows SOC analysts to create dashboards, perform searches, and generate reports based on the security data collected by Wazuh [72].

Figure 2.8 details every component of the Wazuh solution that leveraging the ELK stack and OSSEC provides a robust solution with many of the aspects that are expected in

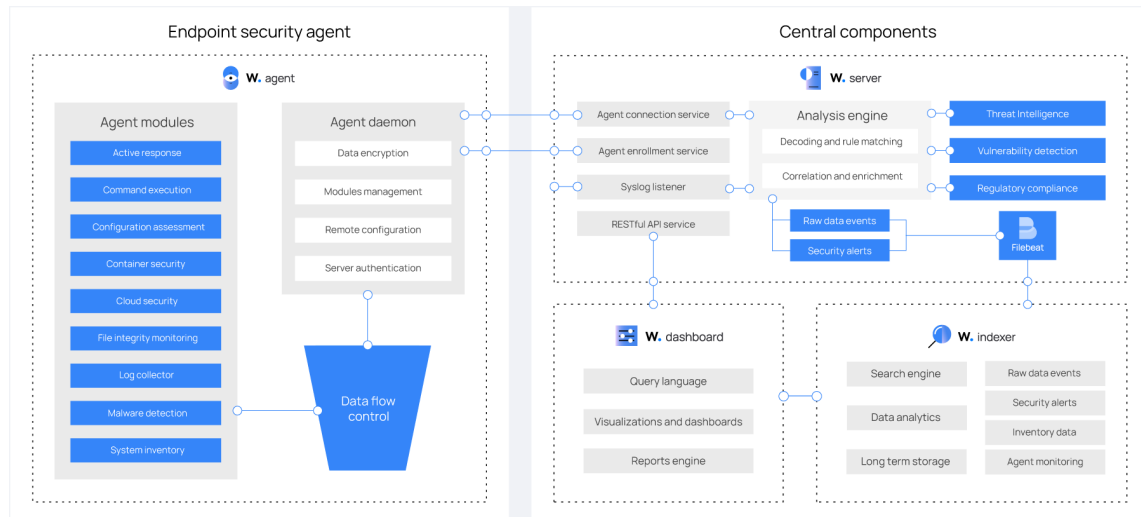


FIGURE 2.8: Wazuh Dataflow Architecture [70]

a modern endpoint security solution and coupled with the data visualization needed for SOC teams.

2.8 Security Operations Center

A Security Operations Center (SOC) is a dedicated team and facility established by an organization to defend against cyber threats and attacks. The main goal of a SOC is to safeguard the organization's crucial assets, encompassing data, systems, and networks, and to ensure their confidentiality, integrity, and availability. It serves as the frontline defense, continuously monitoring the digital infrastructure to identify and neutralize potential security risks while providing context to SOC analysts and minimizing the number of false positives [73].

In today's cyber threat landscape, organizations face numerous risks, including data breaches, ransomware attacks, insider threats, and sophisticated nation-state attacks. In September 2022, Statista released a report highlighting the financial impact of data breaches in the United States and globally. According to the report, the average data breach cost in the United States increased to 9.44 million dollars from the previous year's 9.05 million dollars. Furthermore, the report revealed that the global average cost per data breach in 2022 was 4.35 million dollars. This cost increase reflects organizations' growing challenges in safeguarding sensitive information [74].

The value of a SOC lies in its proactive defense capabilities, which reduce the time gap between detection and response. Several essential factors emphasize the criticality of a SOC:

1. **Early Threat Detection and Response:** With real-time monitoring and advanced threat detection capabilities, a SOC can detect potential threats early, allowing for swift and effective response measures. This timely intervention can prevent attackers from escalating their activities and causing more damage.
2. **Reduced Downtime and Losses:** By promptly identifying and containing security incidents, a SOC helps reduce downtime, minimizing the impact on business operations and preventing potential financial losses.
3. **Compliance and Regulatory Requirements:** Many industries are subject to strict regulatory frameworks governing data protection and cybersecurity. A well-functioning SOC is crucial in meeting compliance requirements and demonstrating the organization's commitment to security.

4. Protection of Reputation and Trust: A successful cyberattack can severely damage an organization's reputation and erode customer trust. A SOC's proactive defense measures protect sensitive information, reinforcing customer confidence in the organization's ability to secure their data.
5. Incident Analysis and Learning: SOC teams conduct thorough post-incident analysis, allowing organizations to learn from security breaches and improve their cybersecurity posture.

Reducing the number of false positives is essential to ensuring that SOC analysts can focus on genuine threats. Using context and advanced analytics helps identify real threats accurately, improving the SOC's efficiency and effectiveness. By integrating context and reducing false positives, a SOC enhances its threat detection capabilities and ensures that analysts can prioritize and respond more effectively to legitimate threats [73].

Chapter 3

Proposed Architecture

This chapter details the framework, configurations, and decision-making process for training and testing an anomaly detection model using syscalls as its primary data. It also describes how this model can be used in a real situation and how it can generate and send an alert to the Open-Source platform and thus alert the SOC team. Finally, it explains how the XAI framework was deployed to help the SOC team interpret the alert, making this application a novel framework use.

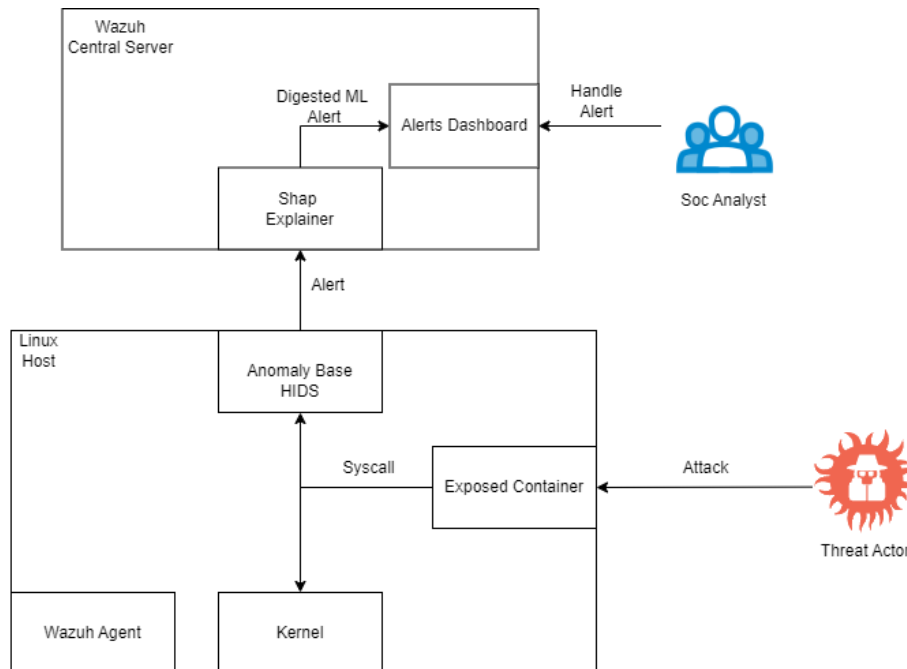


FIGURE 3.1: Architecture proposed for Thesis.

Figure 3.1 illustrates the high-level design of the architecture implemented in this thesis, where the decision engine will evaluate the syscalls of exposed containers, and if a detection is made, the alert is sent to the Explainer, where it will process the anomalous

trace, and finally, it will trigger a rule in the Wazuh, creating an alert in the dashboard of the solution.

3.1 Hardware and Software requirements

To create a prototype of a HIDS with anomaly detection capabilities and the following libraries and resources were needed as can be seen in Appendix A. For the hardware, a virtual machine was used on a cloud provider with the following specifications in Table 3.1.

Operating System	Cpu Type	Number of Virtual Cores	RAM Size
Ubuntu 18.04 Server	Intel Cascade	32	120 GB

TABLE 3.1: Hardware Used to train the Model

Following the framework recommendations, all software development for this project was based on the Python 3.7 version.

For the SIEM, the Wazuh manager was installed in an Ubuntu 20.04 Virtual Machine using version 4.8.1. The Wazuh agents were also installed in the host. All of the installation processes were done following Wazuh's official guide.

3.2 Anomaly Detection Model

This section will describe the steps to create the anomaly detection engine for the proposed HIDS architecture in Figure 3.1, including how the LID-DS was implemented, what data process steps were taken, and how the Decision Engine was trained and tested.

3.2.1 Framework

The researchers that created the LID-DS also created a publicly available Python framework [75]; the framework helps speed up the use of the dataset and can also be used to create your own dataset by leveraging docker and scripts to create other syscall trace and create your own capture. For this thesis, only two sections of the Framework were used, given that the objective of this project is to practically implement an ML model with an already established dataset. The two sections used were:

1. **Algorithms** In this section, there are multiple implementations of algorithms used in HIDS design, such as algorithms for feature engineering, Word Embedding, int Embedding, DBscan, n-gram, and many more. There are also algorithms for decision engine implementations such as LSTM, STIDE, SOM, and others, also some already constructed IDS implementations using ADFA-LD and LID-DS and implementing the aforementioned preprocessing and decision engine algorithms in the context of training an IDS and evaluating the model trained. This framework section allows a quick and clear implementation of many machine learning algorithms, making it accessible to even the less well-versed in the field to implement then.
2. **Dataloader** This section allows the ease of use in the loading of the syscalls traces of the datasets into machine learning algorithms to train, validate and test the models. There are 3 native implementations to load data from the datasets: LID-DS 2019, LID-DS 2021, and ADFA-LD, also in this section, it is possible to determine how to distribute the data of the datasets between the training, validation, and testing phases.

The framework was extensively used in implementing this project. However, many modifications to the source code were made during the project to better accommodate the necessities faced; the major changes will be highlighted in the following sections.

3.2.2 Dataset

As was previously illustrated, the LID-DS is a modern Intrusion Dataset created by researchers at Leipzig University; where it contains traces of syscalls in a relatively modern Linux distro, Ubuntu 18.04, compared to other publicly available datasets, and it contains a lot of contextual information of the syscalls in each trace, different from the older Datasets especially ADFA-LD, the most used Dataset in HIDS researches, that only contains the IDs of the syscall in the traces. The differences between the publicly available Linux datasets can be seen in figure 3.2.

Given that the LID-DS has much more contextual information to train the decision engine and has relatively new and modern attacks, it was chosen as the Dataset to train the model of this project. However, it does not have many research papers that attest to its credibility, and it also does not contain any known practical applications documented, this being a novel use of the dataset that what is known, that is used most of the time as

KDD	UNM	ADFA-LD
open(2): read system call open(2) event-ID 72 AUE_OPEN_R event class fr(0x00000001) audit record: header token, path token, [attr token], subject token, return token	# PID Syscall, ... 162 4, 162 2, 162 66, ...	# Syscall Syscall ... 54 175 120 ...

NGIDS-DS							
DATA,	TIME,	PID,	PATH,	Syscall,	Event ID,	Categ.,	Subcat, Label
11/03/2016,	2:45:01,	1830,	/sbin/upstart-dbus-bridge,	142,	45354,	normal,	normal, 0
11/03/2016,	2:45:06,	1804,	/bin/dbus-daemon,	256,	45352,	normal,	normal, 0

FIGURE 3.2: Syscall Information on other Datasets.

only a Data engineer study. Also, it is important to state that some infrastructures still use Ubuntu 18.04 on their premises, so the model’s applicability does not stay on a theoretical plane and can have some implementations in real infrastructures. Figure 3.3 illustrates the traces in the LID-DS.

```

TIME CPU PROCESS (PROCESS ID) ENTER(>)/EXIT(<) SYSCALL ARGUMENTS
...801301340 1 mysqld (5940) > recvfrom fd=41(<4t>127.0.0.1:54926->127.0.0.1:3306) size=102
...801302094 1 mysqld (5940) < recvfrom res=102 data=...mysql_native_pass... tuple=NULL
...801305987 0 apache2 (6264) < sendto res=106 data=...mysql_native_passw...
...801309746 1 mysqld (5940) > sendto fd=41(<4t>127.0.0.1:54926->127.0.0.1:3306) size=48 tuple=NULL
...801309909 0 apache2 (6264) > poll fds=12:431 timeout=1471228928
...801320710 1 mysqld (5940) < sendto res=48 data=,...mysql_native_password.jrz1$^/1VGCWk5cwJL! [.

```

FIGURE 3.3: Syscall traces of LID-DS 2019.

The LID-DS has two available versions: 2019 and 2021. Due to hardware limitations, the 2019 version was chosen, given that the 2021 version is six times bigger and would require extensive data processing to avoid bias. So, when the project refers to LID-DS, it is referencing the 2019 version, not the 2021.

The LID-DS is divided into 10 scenarios, as can be seen in figure 3.4 where the researchers captured normal traces and attack traces; it is important to point out that both traces were synthetically generated, where common CVEs and CWEs were explored, and the syscalls were captured. All of the research about this dataset used each scenario separately to evaluate their anomaly models, and even the framework itself only has the built-in function to load the data from each scenario separately. However, given that this thesis aims to build a holistic anomaly detection that might detect zero-day attacks, using only one scenario to train the detection model is not ideal. Thus, instead of using each scenario separately, the scenarios were concatenated into one scenario.

In doing so, the final dataset has the following characteristics: 10142 normal traces, 1142 attack traces, and 93 unique syscalls. Also, to avoid bias in the training, the dataset

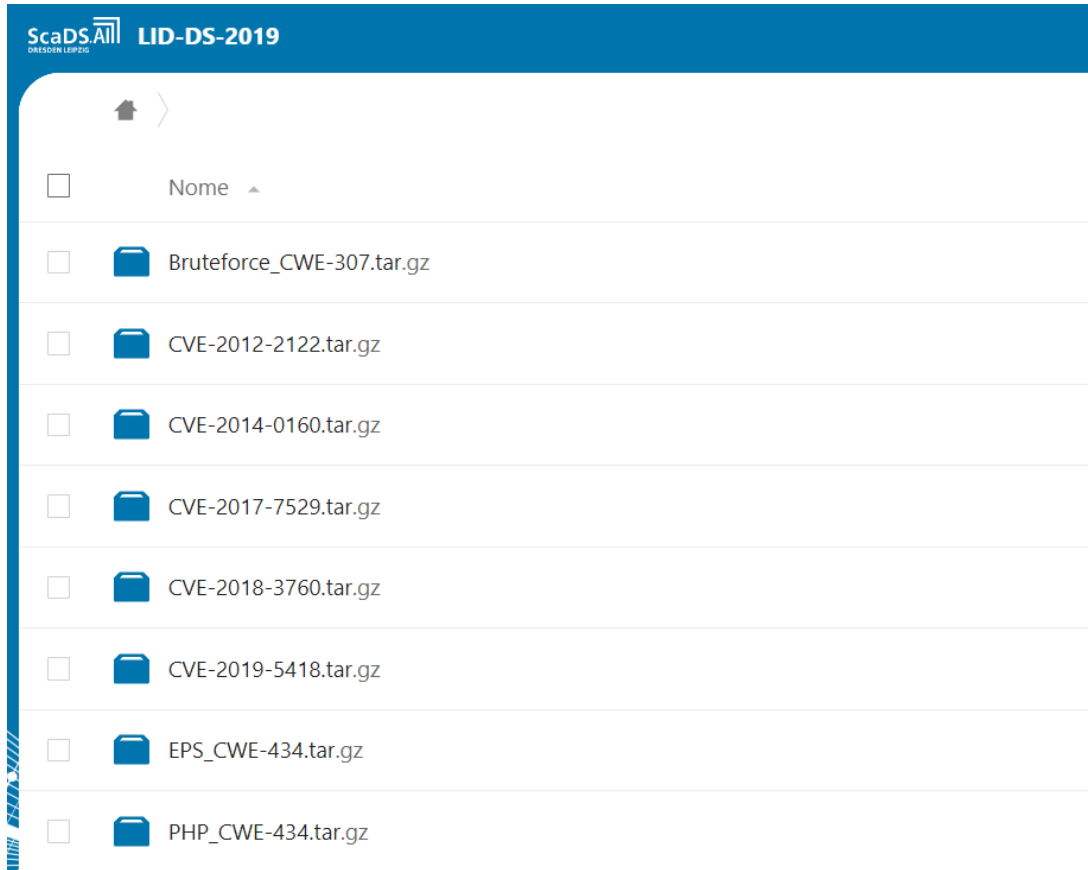


FIGURE 3.4: LID-DS is Original Layout

was randomly shuffled to avoid long sequences of one scenario being represented.

3.2.3 Data Preprocess

The data preprocessing phase has two main stages. The first is to balance the data and allow the network to train optimally without bias. The second is to create features that the decision engine can use to ingest to train its parameters.

The model to be trained will use the unsupervised approach. The goal is for the model to learn normal patterns, and any significant deviation from these patterns during testing will be flagged as an anomaly. Thus, it creates a baseline of normal behavior and detects any deviation.

The previous section detailed the distribution of the unified LID-DS dataset that will be used to train the detection model; being extremely unbalanced, too many normal traces compared to the attack traces, a normal characteristic in HIDS datasets, so better leverage the dataset and due to hardware limitations, the size of the LID-DS made the use in its entirety impossible with the current set up of the machines available for the thesis, the

technique of randomly undersampling of the normal traces and attack traces was applied, so now the data to be used in training has 1100 normal traces and 350 attack traces. 70% of the normal traces will be used in the training of the network, 5% will be used in the validation phase, and the remaining normal traces and 100 of the attack traces will be used in the testing phase of the model.

Given that each scenario has a different number of traces, as shown in 2.2, two approaches were used to create the model. The first was using only random traces to create the datasets, not worrying about the distribution of the scenarios and thus being susceptible to representing a specific scenario. The second approach is described in the method below. To avoid bias in the results, the traces of each scenario were evenly distributed for the training, validation, and testing datasets. The chosen method will be explained in the following section.

```
1 def _shuffle_and_partition_data(self):
    # Shuffle the normal recordings
3     for image in self._normal_recordings:
        random.shuffle(self._normal_recordings[image])
5
        print('Normal recordings:', sum(len(records) for records in self.
            _normal_recordings.values()))
7     print('Exploit recordings:', len(self._exploit_recordings))
9
    num_images = len(self._normal_recordings)
    normal_training_size_per_image = 1000 // num_images
11    normal_validation_size_per_image = int(normal_training_size_per_image * 0.05)
    normal_test_size_per_image = int(normal_training_size_per_image * 0.1)
13
    training_normal = []
15    validation_normal = []
    test_normal = []
17
    # Evenly partition normal recordings by image
19    for image in self._normal_recordings:
        recordings = self._normal_recordings[image]
21        if len(recordings) >= normal_training_size_per_image +
            normal_validation_size_per_image + normal_test_size_per_image:
```

```

        training_normal.extend(recordings[:normal_training_size_per_image])
23         validation_normal.extend(recordings[normal_training_size_per_image:
normal_training_size_per_image + normal_validation_size_per_image])
        test_normal.extend(recordings[normal_training_size_per_image +
normal_validation_size_per_image:normal_training_size_per_image +
normal_validation_size_per_image + normal_test_size_per_image])
25     else:
        print(f"Not enough recordings for image {image} to meet the required
partition sizes.")
27
    # Shuffle and limit exploit recordings ensuring each image is represented
29    exploit_recordings_by_image = {}
    for exploit in self._exploit_recordings:
31        image = exploit.metadata()['image']
        if image not in exploit_recordings_by_image:
33            exploit_recordings_by_image[image] = []
            exploit_recordings_by_image[image].append(exploit)
35
    exploit_test_data = []
37    for image, exploits in exploit_recordings_by_image.items():
        print(f"Image {image} has {len(exploits)} exploit recordings")
39        random.shuffle(exploits)
        exploit_test_data.extend(exploits[:10]) # ensuring we get an equal
distribution if possible
41
    # Combine normal and exploit recordings for test set
43    self._training_data = training_normal
    self._validation_data = validation_normal
45    self._test_data = test_normal + exploit_test_data

47    random.shuffle(self._training_data)
    print('Training data:', len(self._training_data))
49    random.shuffle(self._validation_data)
    print('Validation data:', len(self._validation_data))
51    random.shuffle(self._test_data)
    print('Test data:', len(self._test_data))

```

LISTING 3.1: Dataloader used to distribute the scenarios for the training of the model.

For the feature-engineer it was chosen the modern approach of natural language processing of Word embedding, where Word embedding converts words, or in this case, syscalls, into dense vector representations, where similar words or syscalls are mapped to nearby points in a high-dimensional space. Instead of treating syscalls as isolated events, embeddings capture their relationships based on the surrounding context in sequences of system behavior [16]. Techniques like Word2Vec or FastText learn these vectors by training on syscall co-occurrences, identifying patterns where specific calls often occur together. This allows intrusion detection systems to recognize similar malicious behavior patterns, even when attackers use slightly different syscalls. The dense, context-aware representation enables better anomaly detection, reduces computational complexity, and improves the systems ability to generalize to new or evolving attack techniques. The approach was chosen because it is used in modern studies, which entails better results in the evaluation phase of the models, especially for deep learning decision engines. However, as pointed out at [50] in the testing phase, it might create a false sense of how good the model is. The Python library of Word2Vec was used to create the model for generating the features, and to use it, an integer dictionary was made of all of the distinct syscalls (93). Then, the word embedding model was trained on vector size four and window size four in 5000 epochs; these sizes are relatively small compared to other applications of Word2Vec. However, due to the small size of the dictionary, a larger vector and window size would overfit the model, and all the contextual information that Word2vec provides would be lost. After that, an N-gram of said embeddings is created to expand the semantic relationship.

Other values for the W2Vec embedding were tested. However, the combination of Vector size four and Window size four yielded the best results, and thus, these values were used for the rest of the project.

At the end of the process, each syscall to train the model will have a constructed feature of size 16 that will be used to train and test the Neural Network. The syscalls below illustrate the behavior of the feature extraction process.

```
#syscall_3: 18117 22:25:03.660625616 0 0 VM 2248 < futex res=0
#Feature_3: (39,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812,
-0.9036383032798767, 0.35354042053222656, 0.22540293633937836, 2.5172526836395264,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812)
#syscall_2: 18116 22:25:03.660624012 0 0 VM 2248 >
futex addr=7F162C0CA128 op=129 val=1
```

```
#Feature_2: (39,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812,
-0.9036383032798767, 0.35354042053222656, 0.22540293633937836, 2.5172526836395264,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812)
#syscall_1: 18115 22:25:03.660614563 0 0 VM 2248 < futex res=-110(ETIMEDOUT)
#Feature_1: (39,
-0.9036383032798767, 0.35354042053222656, 0.22540293633937836, 2.5172526836395264,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812,
-8.343196868896484, 8.303723335266113, 0.3890305161476135, 1.053705096244812)
```

LISTING 3.2: Syscall captured with its respective feature to be evaluated in the LSTM

Whereas the *futex* syscall has an integer mapped to it, 39, and the entire syscall generates a 16 vector of weights as features, it is essential to note that LID-DS allows the possibility of developing different features based not only on the syscall ID but many other factors of the syscall leveraging the Word Embedding capabilities.

3.2.4 Decision Engine

3.2.4.1 Training

The decision engine chosen was the LSTM Recurrent Network, given that research shows that it can be resilient against obfuscation techniques [26] and its capabilities in capturing temporal dependencies and handling complex data. Thus, it also can leverage the LID-DS rich syscall parameters and handle possible data sparsity in the dataset [61]; however, given that most of the consolidated studies used ADFA-LD or older datasets to test the LSTM usability in anomaly detection for HIDS, the project here presented will show the effectiveness of the model using a very recent and rich dataset in a controlled environment, this condition still do not represent real-world data, but it can better leverage real situations that the older Datasets did not mimic.

One of the downsides of training an LSTM is that it is extremely resource-expensive. Given the project's limited budget, the hardware was limited, and thus, a sample of the Dataset was used to create the Model. To create the LSTM, the following parameters were used:

1. *Hidden Layers* The hidden layer refers to the depth of the network, i.e., the number of LSTM layers stacked on top of each other. More hidden layers can enable the model to learn more abstract representations of the data, which can be beneficial for

Hidden Layers	Hidden Dimensions	Batch Size	Epochs
1	128	160	11

TABLE 3.2: Values used in the creation of the LSTM

capturing long-range dependencies and hierarchical features. However, this also increases the computational cost and the likelihood of vanishing or exploding gradients, making training more challenging. With the limited resources, fairly limited resources, the hidden layers were always maintained as 1, given that the training host could not take the workload in the first attempt to use other values.

2. *Hidden Dimensions* The number of units in the hidden layer determines the size of the hidden state vector and cell state vector within the LSTM cell. A larger hidden dimension allows the model to capture more complex patterns and dependencies in the data, potentially improving performance on complex tasks. However, it also increases the computational complexity and the risk of overfitting, especially with limited training data. After some fine-tuning and testing, it was placed with 128 as its value.
3. *Epoch* An epoch is one complete pass through the entire training dataset. More epochs allow the model to learn more from the data, potentially improving performance as the training progresses. However, too many epochs can lead to overfitting, where the model learns the training data too well and performs poorly on unseen data. The 11 Epochs are relatively standard for training LSTM and were the ones used in the project.
4. *Batch Size* e batch size is the number of training samples processed before the model's internal parameters are updated. A smaller batch size offers a more fine-grained update to the model parameters, which can improve the model's ability to generalize. Still, it also increases the variance of the gradient updates, potentially leading to noisier training. Conversely, a larger batch size provides more stable and accurate gradient estimates but requires more memory and may lead to less frequent updates, slowing the convergence. The number of 160 used was after a long interaction of model overfitting or not generalizing enough.

3.2.4.2 Testing

As stated in the previous section, two methods of sampling the data were evaluated: the first was truly random, and the second randomly chose a trace in each scenario to avoid overrepresenting a scenario in the model's training data. After the model was trained, the framework has the method `detect`, as seen in listing 3.3, that allows the testing of the model using the test data that was separated, as seen in listing 3.1.

```

1  def detect(self) -> Performance:
    """
3      detecting performance values using the test data,
        calling performance object for measurement and
5      plot object if plot_switch is True
    """
7      data = self._data_loader.test_data()
        description = 'anomaly detection'.rjust(27)
9
        for recording in tqdm(data, description, unit=" recording"):
11            self.performance.new_recording(recording)
                if self.plot is not None:
13                    self.plot.new_recording(recording)
15
                for syscall in recording.syscalls():
                    is_anomaly = self._final_bb.get_result(syscall)
17                    self.performance.analyze_syscall(syscall, is_anomaly)
                        if self.plot is not None:
19                            self.plot.add_to_plot_data(anomaly_score,
                                                                syscall,
21                                                                self.performance.get_cfp_indices())
23
                self._data_preprocessor.new_recording()
25
                # run end alarm once to ensure that last alarm gets saved
                if self.performance.alarms is not None:
27                    self.performance.alarms.end_alarm()
        return self.performance

```

LISTING 3.3: Method to Assess the model effectiveness using the test data.

After extensive tuning of the model's parameters, these were the performance results of the models for each data sampling can be seen in the following table:

Type of Training Dataset	Detection Rate	Precision	FAR
Truly Random	81.303%	85.290%	14.022%
Evenly Distributed	72%	98.630 %	1%

TABLE 3.3: Metrics for Anomaly models

The method chosen for the rest of the project was the evenly distributed scenarios for the training data. However, the detection rate of the truly random approach is higher, given that the objective of the project is to create a tool that will be used in a real-world context, an anomaly detector with this high of a FAR and with lower precision as the truly random, makes not suitable for a security operations team, giving that alert fatigue might create scenarios where the alerts are not giving the proper treatment.

3.3 Host Evaluation

With the Anomaly Detection Model now able to assess and detect possible attacks, the next step in the project is to use the model in a real-world situation with the exploit of CVEs that did not exist in the period of the creation of the dataset and thus proving the capabilities in detecting Zero Day attacks, especially in scenarios where remote code execution is achieved, giving that it gives the threat actor possibilities in ownership of the host and thus capabilities of bypassing further security controls.

To do so, proper endpoint protection software will not be created for this solution. There are other aspects to be designed, such as tamper-proofing, creation of known signatures, file monitoring, and many other aspects that are not in the scope of the project, being that the main focus is to evaluate if the model can detect new attacks and does not generate an extensive amount of False Positive, making it unusable in a security operation context.

As previously stated, the LID-DS was created using container syscalls; thus, the model will first be tested by monitoring a single vulnerable container, where it will collect the syscall of the container and after that, it will evaluate in the model if the syscall is anomalous. Later, the architecture will be tested for the entire host, where the complexity is much greater, thus putting into perspective if the LID-DS, although containing detailed syscall

parameters, is truly capable of reproducing the complexity of a host or syscall containers can only be used in creating a model that detects attacks on containers.

So, the only need of the architecture to assess this information is to collect the syscall on a live system and then pass it through the model so that an alert can be generated. To do so, the LID-DS repository was again leveraged, given that they created a code base that monitored containers and used sysdig and STIDE Ngram to create a model to detect anomalies; the leveraged code base for this project was the backend section where there was already a logic of collecting syscalls and running through a model [76]. The major modifications in the code were eliminating most of the frontend logic, given that Wazuh would be used as the alert aggregator, and converting the N-gram evaluation logic to a per syscalls analysis. All of the training processors for W2V embedding and the model were saved to be used in the live host.

The code creates two threads, one for observability of the syscall and the other to save the syscall in a queue for evaluation. In Chapter 4, it will be shown that for evaluating the detection model, syscalls of different containers and from the host itself will be collected, and to do so, the following method is used.

```
def __init__(self, data_handling, pid):
2     # Initiate syscall deque
    self.deque_syscall = collections.deque()
4     # Initiate write deque thread
    self.write_thread = threading.Thread(
6         target=self.write_syscalls, args=([pid]))
    self.write_thread.start()
8     # Initiate read deque thread
    self.read_thread = threading.Thread(
10        target=self.read_syscall, args=([]))
    self.read_thread.start()
12    # Initiate data_handling
    self.data_handling = data_handling
14
def start_sysdig_and_read_data(self, pid):
16    self.sysdig_process = None
    try:
18        print("Sysdig monitoring Node App")
        pid_argument = "container.id=76514ffe2de8"
```

```
20     formatting = "-p %evt.num %evt.time %evt.cpu %user.uid %proc.name %thread  
    .tid %evt.dir %evt.type %evt.args"  
    # listen node process  
22     sensor_command_line = ['sudo', '/usr/bin/sysdig', pid_argument,  
                             formatting]  
24     self.sysdig_process = subprocess.Popen(  
        sensor_command_line,  
26        stdout=subprocess.PIPE,  
        encoding="utf-8")  
28     yield self.sysdig_process  
    finally:  
30     self.sysdig_process.terminate()  
    self.sysdig_process.kill()
```

LISTING 3.4: Sysdig Capture Code

Leveraging the sysdig tool, the program can parse exactly the syscall format used in creating the LID-DS and training the Model. Since the sysdig tool uses eBPF technology, the collection is reliable and fast.

With the collection in place, each syscall is transformed into the feature engineered in the previous section. This feature is passed through the model and receives a score to determine whether it is an anomaly.

The following logic was applied to create an alert: in a window of eleven syscalls observed, an alert is generated if seven are considered anomalies. This logic avoids false positives if a single syscall is detected as malicious, creating a more robust detection algorithm. These numbers were chosen based on the n-grams used in multiple research, given that these prime numbers are the most commonly used in machine learning exercises.

Three major lists are used in the code. The first is a binary list that keeps track of the distribution in the 11 syscall window of the number of syscalls detected as malicious. The other two are lists of the syscalls and their features. These two lists are used as the contents of the Alert that will be sent to the Wazuh instance.

3.4 SIEM Integration

Working as the SIEM for this architecture, Wazuh has two main attributions: it receives alerts from its native agent, which will be installed on the monitored host/container to

create a baseline to see if the agent, with its signature capabilities, can detect the attacks, and it also receives the custom machine alerts from the implementation of this project. Thus, it can validate the importance of such observability of the syscalls in a HIDS, not only for anomaly detection but also for creating signatures.

3.4.1 Explanation Model

SHAP is an Explainable Artificial Intelligence (XAI) method that helps interpret model predictions by assigning importance or "weights" to each feature used in a decision. In anomaly detection, SHAP can explain how features contribute to identifying abnormal behavior by generating Shapley values. These values quantify each feature's contribution, making it possible to understand why the model flagged a particular instance as anomalous.

By calculating the marginal contribution of each feature, SHAP provides insights into which aspects of the input data are most influential for the model's decisions. In this context, SHAP is particularly useful for models like LSTMs (Long Short-Term Memory), where understanding temporal dependencies can be challenging. SHAP assigns a weight between -1 and 1 to each feature, helping to clarify which features were pivotal in detecting the anomaly. The closer the SHAP value is to 1, the more substantial the feature's contribution to the anomaly decision.

For this thesis, SHAP was chosen because it can complement the feature vectors generated during training by highlighting the role of each feature in the anomaly detection decision.

The Python SHAP library provides an accessible interface with multiple implementations for different models, allowing complex calculations to happen seamlessly in the background. A function from the DeepExplainer class was utilized to create the explainer for the LSTM model used in this work, as seen in the Listing 3.5. The explainer uses the results of the trained model, in conjunction with the input feature vectors, to generate SHAP values.

```
x_tensors_path = '/home/hleo/Explainer/training_x_tensors.pt'
2 model_path = '/home/hleo/LID-DS/algorithms/Models/All_scenarios/LSTM/
    hid128taTrueng4-emb4-rvFalse-tdFalse-tcfFalse.model'
lstm = (torch.load(model_path)).to("cpu")
4 x_tensors_final = torch.load(x_tensors_path).to("cpu")
```

```
lstm_wrapper = LSTMWrapper(lstm, lstm.hidden_dim, lstm._hidden_layers)
6 explainer = shap.DeepExplainer(lstm_wrapper, x_tensors_final[:100])
```

LISTING 3.5: Use of the SHAP library to create the Explainer.

Once the explainer is created, it computes SHAP values for the 16 features in the input vector, representing a syscall. Each SHAP value indicates how much a specific feature contributed to the model's anomaly detection decision. The SHAP values range from -1 to 1, where values closer to 1 suggest a stronger influence on the anomaly detection.

However, the anomaly detection alert system uses an 11 n-gram of syscall features, requiring 7 out of 11 to be flagged as anomalous to trigger an alert. If SHAP values for all 11 syscalls were reported separately, this would result in an overwhelming number of graphs and potentially confusing data for the analyst.

To avoid complexity, a routine was developed to aggregate the SHAP values of all 11 syscalls. This routine averages the SHAP values, presenting the analyst with a single, clear graph that represents the contribution of all syscalls involved in the alert.

Figure 3.5 represents the result provided by the explainer in the architecture, where it displays what features had the biggest or least impact on the decision and the SHAP values associated with each feature.

3.4.2 Wazuh Alert

As previously explained in the Figure 3.1, the detection model instance in each node does not communicate directly to the Wazuh manager. The explainer code is the one that receives the models' communications and creates an alert for the Wazuh Manager. A simple custom rule was created in the OSSEC configurations of the Wazuh to create the anomaly alert.

```
<group name="custom-rules">
  <rule id="220526" level="14">
    <description>Anomaly detected from syslog</description>
    <match>Anomaly detected:</match>
  </rule>
</group>
```

LISTING 3.6: Rule Created in Wazuh Manager to generate the anomaly alert.

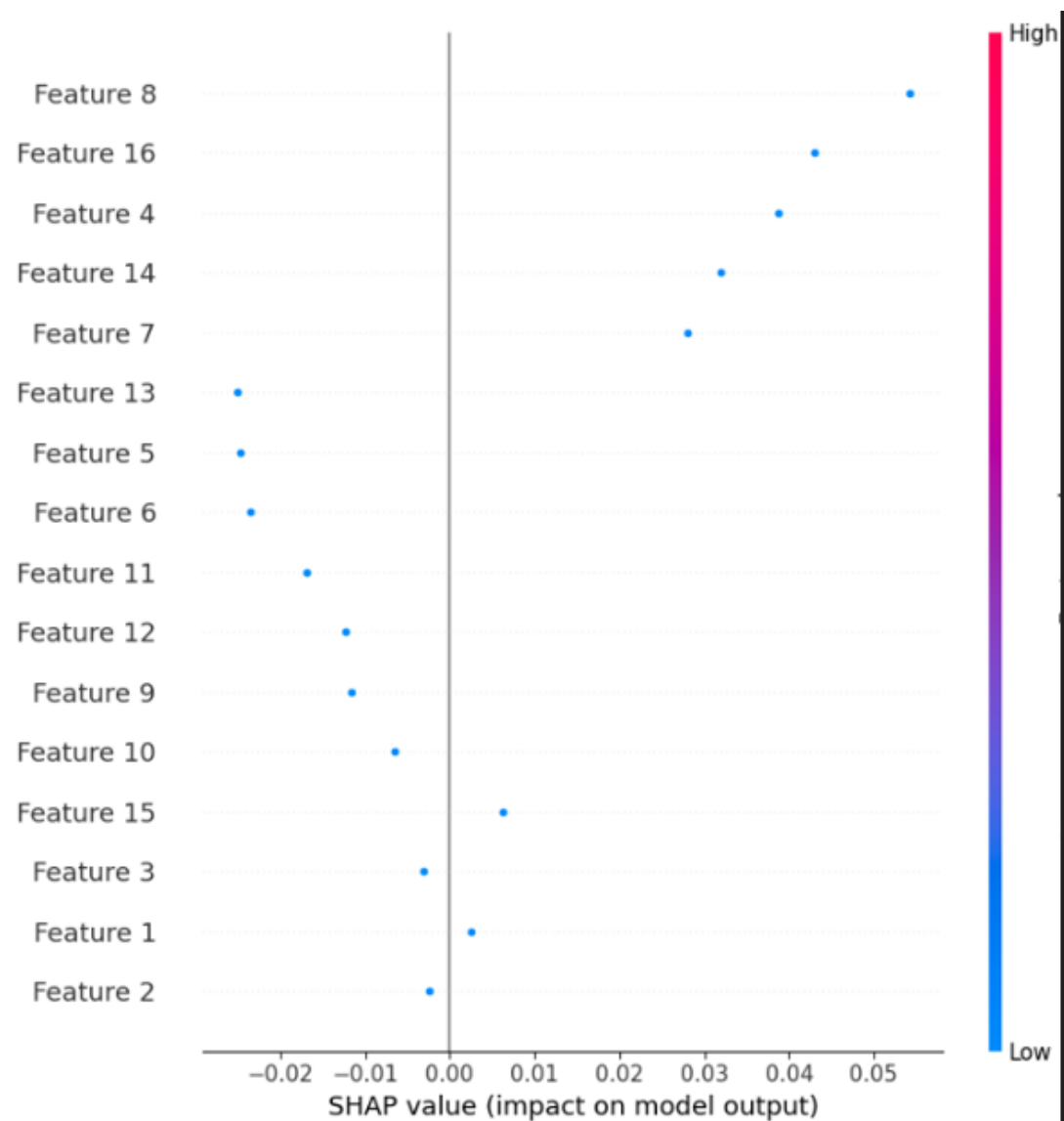


FIGURE 3.5: Example of output of the Explainer

No other major modifications were made to Wazuh's configurations or the prebuilt OSSEC in the agent, so all of the alerts that might appear are the out-of-the-box configurations that the open source provides.

3.5 Concerns

Given that the project did not have the objective of creating a functional and market-ready HIDS, some concerns were not carefully considered during the process of creating the HIDS, such as the resources consumption that the agent presented, using a coding language that is faster and more responsive should be the main concern when developing a HIDS solution, given that the least resource the solution consumes the better. Tampering protection was not implemented since this solution is needed in a real-world scenario to avoid the threat actor disabling the solution and proceeding with the attack.

Also, given that the LID-DS does not have the entire syscall table of Ubuntu 18, syscalls that were not included in the training might trigger alerts that are False positives. This issue will be addressed in Chapter 4, where the model will be tested in real-world scenarios/attacks.

Another possible problem to be evaluated in the next chapter is that, given that the LID-DS was created using traces of containers, models created with the dataset might struggle with overly complex systems. The data captured in containers is not as complex or dense as the one present in a monolithic host.

Chapter 4

Evaluation

This chapter presents a set of use cases to evaluate the architecture performance detailed in Chapter 3. While the tests conducted here do not fully replicate the complexities of real-world HIDS deployments, they showcase the capabilities of an anomaly detection model trained on publicly available datasets. These evaluations provide insights into how such models might enhance infrastructure implementations when applied to real-world attack scenarios.

The chapter is organized into three sections. The first outlines the environment setup, including the architecture configuration and target systems used to validate detection and interpretation. The second section describes the simulation phase, detailing the execution of tests and the detection results. Finally, the third section discusses how the simulation outcomes contribute to the overall conclusions of the thesis.

4.1 Preparation Phase

Five vulnerabilities, each representing a different attack vector, were selected to set up the testing architecture. These vulnerabilities were chosen to assess the explainers ability to differentiate between various attack types and provide helpful feedback to analysts handling alerts. Notably, these vulnerabilities were not present when the LID-DS dataset was created, meaning anomalous behavior should serve as the primary indicator of compromise. Additionally, an additional test will evaluate whether a model trained on container traces can effectively detect intrusions on a monolithic host.

The code was modified to include the container ID when executing the backend logic to capture system calls for only the containers running the vulnerable applications. To

Application	Vulnerability	Type	Attack Execution
Apache HTTP Software	CVE-2021-42013	Remote Code Execution	Metasploit Module
Apache Spark	CVE-2022-33891	Remote Code Execution	Metasploit Module
Apache Commons Text	CVE-2022-42889	Remote Code Execution	Custom Payload
Apache Commons Text	CVE-2022-42889	Server Side Request Forgery	Custom Payload
PHPGurukul Hospital Management System	CVE-2023-7172	SQL Injection	Simple SQL Injection Command
OURPHP	CVE-2023-30212	Cross Site Scripting	Simple XSS payload

TABLE 4.1: Scenarios tested for intrusion detection in containers

extend the model to the entire system, the container ID argument was replaced with the host argument in the Sysdig program, enabling it to capture all syscalls on the host.

The table 4.1 represents all of the Applications tested, the associated vulnerabilities, and how the exploitation will be done.

Finally, a simple attack will be conducted to evaluate the model's behavior, monitoring the entire host syscalls. An already pre-compile binary will be brought into the system and executed. Thus, the capabilities of detecting attacks in a host with a much more complex syscall pattern than a container will be attested.

All of the elements in the architecture were contained in the same VLAN, and all traffic was allowed to avoid any interruptions in the tests; also, for the container tests, the attacks are made by an external source, and to evaluate the host, the command will be made by an authenticated user inside the host, thus creating the necessity as the last line of defense for the HIDS.

4.2 Simulation Phase

4.2.1 Container Evaluation

4.2.1.1 Container Normal Behavior

A simple script was created to simulate user interactions by accessing existing and non-existent web pages to test the containers' regular behavior. This script is crucial for evaluating whether any syscalls need to be whitelisted to avoid false positives and to determine whether excluding certain syscalls from monitoring still allows the model to detect anomalies effectively. Additionally, manual tests were conducted on other elements of the containers that the script could not handle.

During testing, the script simulating normal container usage and manual tests triggered some false positive alerts from the model. Based on these alerts, certain syscalls were filtered out from the evaluation process, as shown in the Listing 4.1. While this filtering introduces a potential gap in detection coverage, it is a necessary trade-off to prevent overwhelming the SOC team with unnecessary alerts.

```
def filter_syscall(self, parsed_syscall):
    """
    return false if syscall is filtered out
    """
    if parsed_syscall[8] == "wait4":
        return False
    if parsed_syscall[8] == "times":
        return False
    if parsed_syscall[8] == "epoll_wait":
        return False
    if parsed_syscall[8] == "epoll_ctl":
        return False
    if parsed_syscall[8] == "getsockname":
        return False
    if parsed_syscall[8] == "getdents64":
        return False
    if parsed_syscall[8] == "fstat":
        return False
    if parsed_syscall[8] == "openat":
```

LISTING 4.1: Syscalls excluded in the evaluation of the containers

Concluding this phase, an external host was set to explore the vulnerabilities of the containers.

4.2.1.2 CVE-2021-42013

The vulnerability CVE-2021-42013 is a flaw found on Apache HTTP software version 2.4.49. An attacker could use a path traversal attack to map URLs to files outside the directories configured by Alias-like directives. If the usual default configuration does not protect files outside these directories "require all denied," these requests can succeed. If CGI scripts are also enabled for these aliased paths, remote code execution could be achieved. [77]. To achieve an environment for exploration, the container used in this POC [78] was set up in the host monitored by the HIDS.

To explore the CVE-2021-42013, the Metasploit module was used, and as can be seen in figure 4.1, the attack was successful, and the external host could echo a command in the container.

```
msf6 exploit(multi/http/apache_normalize_path_rce) > run

[+] echo Exploit realizado com sucesso
[*] Using auxiliary/scanner/http/apache_normalize_path as check
[+] http://192.168.1.248:8080 - The target is vulnerable to CVE-2021-42013 (mod_cgi is enabled).
[*] Scanned 1 of 1 hosts (100% complete)
[*] http://192.168.1.248:8080 - Attempt to exploit for CVE-2021-42013
[*] http://192.168.1.248:8080 - Generated payload: echo Exploit realizado com sucesso
[!] http://192.168.1.248:8080 - Dumping command output in response
Exploit realizado com sucesso

msf6 exploit(multi/http/apache_normalize_path_rce) > █
```

FIGURE 4.1: Attack used in HTTP Apache server

Figure 4.2 represents the container normal usage and syscalls that are normally used by the system, as for 4.3 it represents the syscalls at the moment of the attack and generation of the alert by the model.



FIGURE 4.2: Syscall Traces of regular usage of the Apache HTTP server

After the attack is detected, the model sends the alert information to the Explainer, generates the graphic with the SHAP model, and forwards all of the alert information to the Wazuh instance.

In 4.4, the analysts receive information on which host and container the alert was generated, the generated graph's URL that explains the alert's possible cause, and details

FIGURE 4.3: Syscalls traces of attack and the generation of the alert.

of the syscall trace. With this information, the analyst can make a preliminary analysis of the root cause of the attack.

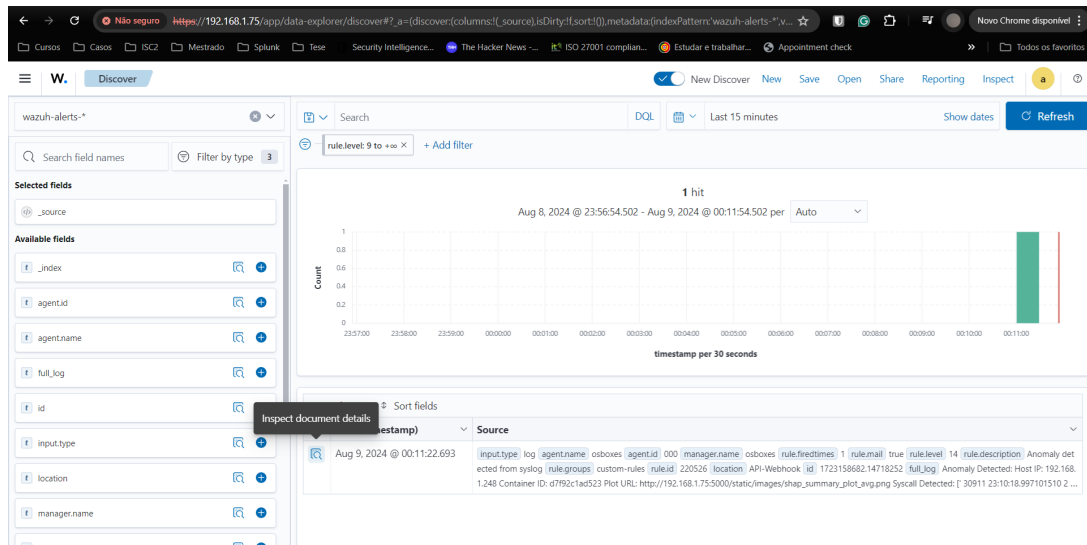


FIGURE 4.4: Wazuh Dashboard with the anomaly detection alert

Figure 4.5 shows what was the syscall trace that triggered the Alert.

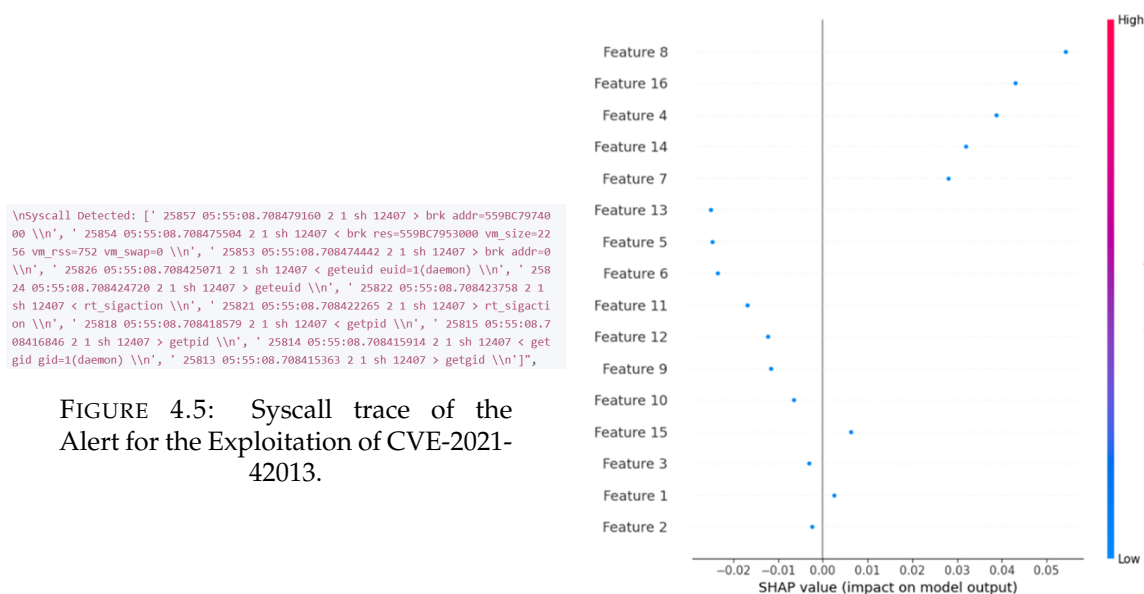


FIGURE 4.5: Syscall trace of the Alert for the Exploitation of CVE-2021-42013.

FIGURE 4.6: Graphic with the Shap Values of the detection.

Figure 4.6 represents each of the 16 features the model created and each shape value that correlates with the possible technique used in the attack. More information regarding these labels will be expanded in the results section of this chapter.

With this simulation, the model could detect an exploration of a vulnerability that it was not trained on, thus showing the anomaly detection system's zero-day capability.

4.2.1.3 CVE-2022-33891

The vulnerability CVE-2022-33891 is in Apache Spark versions 3.1.1 and 3.1.2. The Apache Spark UI can enforce access control lists (ACLs) through the configuration option `spark.acls.enable`. This setting uses an authentication filter to verify a user's permission to view or edit the application. If ACLs are enabled, a code path in `HttpSecurityFilter` may allow impersonation by accepting an arbitrary username. A malicious user could exploit this to access a permission check function, creating and executing a Unix shell command based on the provided input. [79] To set the environment with this vulnerability, it was used the docker script in [80], which provides a vulnerable application.

Figure 4.7 illustrates the attack on the Spark container using another Metasploit module; in this attack, a Python shell can be established in the container, and to test, a command is made to verify the connection.

```
msf6 exploit(linux/http/apache_spark_rce_cve_2022_33891) > run
[*] Started reverse TCP handler on 192.168.1.74:4444
[*] Running automatic check ("set AutoCheck false" to disable)
[*] Checking if 192.168.1.248:8080 can be exploited!
[*] Performing command injection test issuing a sleep command of 10 seconds.
[*] Elapsed time: 10.094780497000102 seconds.
[+] The target is vulnerable. Successfully tested command injection.
[*] Exploiting...
[*] Command shell session 3 opened (192.168.1.74:4444 -> 192.168.1.248:51138) at 2024-08-09 00:25:36 +0100
uname -a

Linux 1b1f3fdd23ac 4.15.0-213-generic #224-Ubuntu SMP Mon Jun 19 13:30:12 UTC 2023 x86_64 GNU
/Linux
^C
Abort session 3? [y/N] y

[*] 192.168.1.248 - Command shell session 3 closed. Reason: User exit
msf6 exploit(linux/http/apache_spark_rce_cve_2022_33891) >
```

FIGURE 4.7: Exploitation of the CVE-2022-33891

For the Explainer output for both RCE attacks, Figure 4.6 and 4.9, it is possible to observe that the SHAP values of the Features and the most influential Features are very similar, even though the syscalls and the traces of the detection, Figure 4.5 and 4.8 are not similar. This result indicates that the model can detect a similarity between attacks, both are RCEs, even though the CVEs and technologies are not the same.

4.2.1.4 CVE-2022-42889

The vulnerability identified as CVE-2022-42889, commonly referred to as "Text4Shell," affects Apache Commons Text, a widely used library for text parsing and manipulation. This flaw arises from the library's support for variable interpolation, where expressions

```

\nSyscall Detected: [' 32123 05:42:02.966914913 1 33 apache2 3695 > writev fd=10(<
4t>192.168.1.84:26511->172.21.0.4:80) size=4069 \\n', ' 32120 05:42:02.966849999 1
33 apache2 3695 < brk res=56019C259000 vm_size=87000 vm_rss=14988 vm_swap=0 \\n',
' 32119 05:42:02.966834041 1 33 apache2 3695 > brk addr=56019C259000 \\n', ' 32118
05:42:02.966739265 1 33 apache2 3695 < mmap res=0 vm_size=87256 vm_rss=15132 vm_
swap=0 \\n', ' 32117 05:42:02.966720081 1 33 apache2 3695 > mmap addr=7F59F78AD0
00 length=9278 \\n', ' 31937 05:42:02.964768379 1 33 apache2 3695 > switch next=92
22(pytho3.7) pgft_maj=8 pgft_min=714 vm_size=87268 vm_rss=14988 vm_swap=0 \\n',
' 31777 05:42:02.964311024 1 33 apache2 3695 < mmap res=140024381886464 vm_size=8726
8 vm_rss=14988 vm_swap=0 \\n', ' 31763 05:42:02.964284057 1 33 apache2 3695 > mmap
addr=0 length=9278 prot=1(PROT_READ) flags=1(MAP_SHARED) fd=11(<f>/var/www/html/hm
s/admin/vendor/modernizr/modernizr.js) offset=0 \\n', ' 31717 05:42:02.964121701 1
33 apache2 3695 < brk res=56019C259000 vm_size=87256 vm_rss=14988 vm_swap=0 \\n',
' 31712 05:42:02.964105944 1 33 apache2 3695 > brk addr=56019C259000 \\n', ' 31612
05:42:02.962115774 3 33 apache2 3696 > switch next=8664 pgft_maj=5 pgft_min=394 vm_
size=87528 vm_rss=10316 vm_swap=0 \\n']",

```

FIGURE 4.8: Syscall trace of the Alert for the Exploitation of cve-2022-33891.

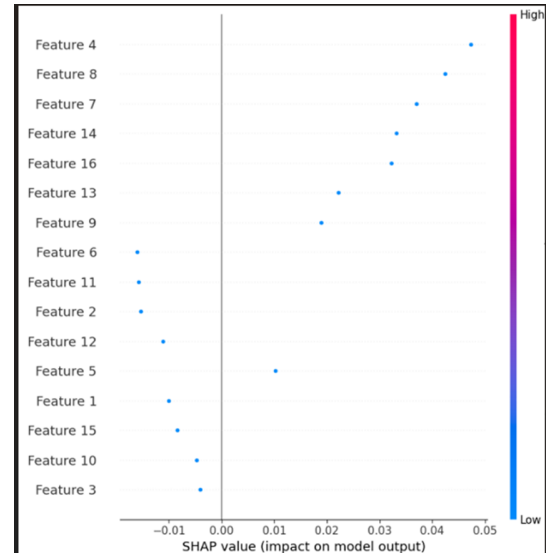


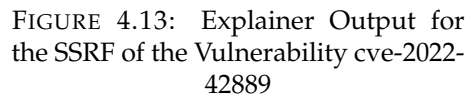
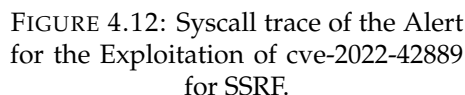
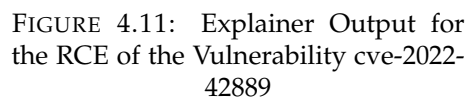
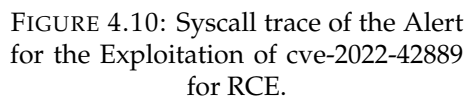
FIGURE 4.9: Explainer Output for the RCE of the Vulnerability cve-2022-33891

in the format "prefix" are dynamically evaluated using instances of the StringLookup interface. Versions 1.5 through 1.9 of the library include default Lookup implementations that can be exploited to perform arbitrary code execution (RCE) or initiate Server-Side Request Forgery (SSRF) attacks. By crafting malicious input strings, attackers can execute code or cause the target system to make unauthorized requests to remote servers. The container the had the vulnerable version of the application was extracted from this repository [81].

CVE-2022-42889 is vulnerable to RCE and SSRF attacks, so both attacks will be tested against the target. A Python script from the repository [82] will be used to exploit the target.

Figure 4.10 shows the trace of the detection. As seen, the command executed by the payload, *touch*, triggered the detection. In Figure 4.11, while the most important feature is not the same, the distribution of the values and other key features are very similar to the RCE examples presented in this thesis.

Figure 4.12 displays the syscall traces of the SSRF attack exploitation. As shown in Figures 4.11 and 4.13, although the detection came from the same container and exploited the same CVE, the techniques used, RCE and SSRF, are different. This results in a completely different set of SHAP values; the most important feature also changes. These results highlight SHAP's contribution in distinguishing between alerts and providing more context to the analyst regarding the specific attack used.



The CVE-2023-7172 is a critical vulnerability in the Admin Dashboard component of PH-PGurukul Hospital Management System version 1.0. This flaw allows remote attackers to perform SQL injection attacks, potentially leading to unauthorized access to the system's database. The vulnerability is considered critical due to the possibility of manipulating or stealing sensitive data. The container hosting the vulnerable application was extracted

from the repository [83].

To exploit the vulnerability, a simple SQL Injection technique was used, as demonstrated in Listing 4.2.

```
admin' -- -
```

LISTING 4.2: Simple SQL Injection snippet

Upon successful execution of the injection query, the following trace was generated in the Alert, as shown in Figure 4.14.

The SHAP values for the explanation output are shown in Figure 4.15, providing insight into the model's interpretation of the event. The SHAP values present in this output are the highest of all testing scenarios. The reason is that SQL Injection is one of the scenarios present in the LID-DS, so the model has a higher confidence in detecting the anomaly.

```
\nSyscall Detected: [' 124140 06:23:08.380039676 1 33 apache2 17889 < poll res=1 f
ds=13:41 \n', ' 123497 06:23:08.364253542 0 33 apache2 17889 > poll fds=13:431 ti
meout=86400000 \n', ' 123488 06:23:08.364239355 0 33 apache2 17889 < sendto res=2
5 data=.....SELECT * FROM users \n', ' 123469 06:23:08.363944448 0 33 apache2 17
889 > sendto fd=13(<4t>172.21.0.4:36878->172.21.0.2:3306) size=25 tuple=NULL \n',
' 123461 06:23:08.363610506 0 33 apache2 17889 < brk res=5555E2A13000 vm_size=8725
6 vm_rss=14640 vm_swap=0 \n', ' 123458 06:23:08.363579529 0 33 apache2 17889 > br
k addr=5555E2A13000 \n', ' 123444 06:23:08.363225410 0 33 apache2 17889 < getcwd
res=24 path=/var/www/html/hms/admin \n', ' 123443 06:23:08.363221213 0 33 apache2
17889 > getcwd \n', ' 123418 06:23:08.362789821 0 33 apache2 17889 < getcwd res=2
4 path=/var/www/html/hms/admin \n', ' 123417 06:23:08.362780684 0 33 apache2 1788
9 > getcwd \n', ' 123407 06:23:08.362554213 0 33 apache2 17889 < recvfrom res=11
data=..... tuple=NULL \n']",
```

FIGURE 4.14: Syscall trace of the Alert for the Exploitation of cve-2023-7172.

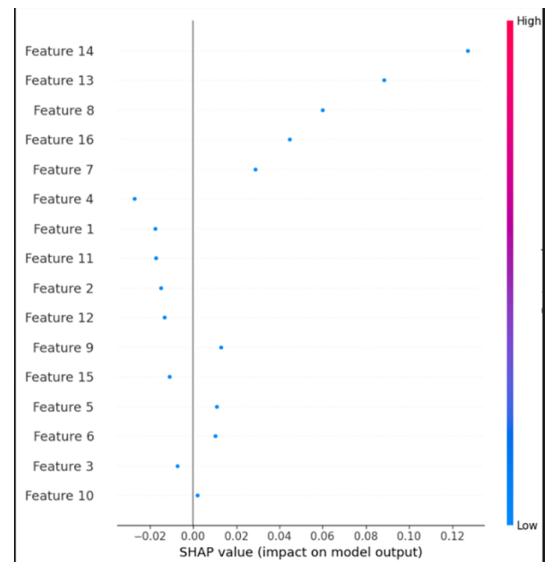


FIGURE 4.15: Explainer Output for the SQLi of the Vulnerability cve-2023-7172.

4.2.1.6 CVE-2023-30212

The vulnerability CVE-2023-30212 in OURPHP <= 7.2.0 is a Cross-Site Scripting (XSS) flaw in the /client/manage/ourphp_out.php file, allowing attackers to inject malicious scripts into the application. These scripts are executed in users' browsers, enabling attackers to steal sensitive data, impersonate users, or perform unauthorized actions. For example, by crafting a malicious URL, an attacker can exploit this vulnerability to compromise

user sessions and gain control over accounts. The container used for this experiment is available in the associated GitHub repository [84].

A simple payload was executed on the host to exploit the vulnerability, as shown in Listing 4.3.

```
/client/manage/ourphp_out.php?ourphp_admin=logout&
out=</script><script>alert("xss")</script>
```

LISTING 4.3: Payload for XSS attack

After successful exploitation, the machine learning model triggered twice, detecting two malicious traces shown in Figures 4.16 and 4.17.

```
\nSyscall Detected: [' 672129 10:55:37.381605065 2 33 apache2 22230 < lstat res=0
path=/var/www/html/client/manage/../../function/ourphp \n', ' 672128 10:55:37.381
601257 2 33 apache2 22230 > lstat \n', ' 672127 10:55:37.381598743 2 33 apache2 2
2230 < lstat res=0 path=/var/www/html/client/manage/../../function/ourphp/Smarty.c
lass.php \n', ' 672125 10:55:37.381554139 2 33 apache2 22230 > lstat \n', ' 6721
24 10:55:37.381546966 2 33 apache2 22230 < getcwd res=28 path=/var/www/html/clien
t/manage \n', ' 672122 10:55:37.381538069 2 33 apache2 22230 > getcwd \n', ' 672
110 10:55:37.381358026 2 33 apache2 22230 < recvfrom res=23 data=....cn|cn|cn.
7..... tuple=NULL \n', ' 672109 10:55:37.381354234 2 33 apache2 22230 > recvf
rom fd=11(<4t>127.0.0.1:55492->127.0.0.1:3306) size=103 \n', ' 672108 10:55:37.38
1353773 2 33 apache2 22230 < poll res=1 fds=11:41 \n', ' 672107 10:55:37.38135285
1 2 33 apache2 22230 > poll fds=11:431 timeout=86400000 \n', ' 672105 10:55:37.38
1340108 2 33 apache2 22230 < recvfrom res=103 data=.....R....def.data.ourphp_web
deploy.ourphp_webdeploy.OP_Webupdate.OP_Webupdate tuple=NULL \n']"
```

FIGURE 4.16: First Syscall trace of the Alert for the Exploitation of cve-2023-30212.

```
\nSyscall Detected: [' 1336830 10:56:35.075831128 1 101 mysqld 22136 > switch next
=0 pgft_maj=0 pgft_min=1 vm_size=1777208 vm_rss=80252 vm_swap=0 \n', ' 671987 10:
55:37.380724815 2 33 apache2 22230 > stat \n', ' 671984 10:55:37.380720887 2 33 a
pache2 22230 < lstat res=0 path=/var/www/html/client/manage/../../function \n', '
671980 10:55:37.380717471 2 33 apache2 22230 > lstat \n', ' 671978 10:55:37.38071
6338 2 33 apache2 22230 < lstat res=0 path=/var/www/html/client/manage/../../funct
ion/ourphp_dz.class.php \n', ' 671976 10:55:37.380707172 2 33 apache2 22230 > lst
at \n', ' 671973 10:55:37.380705518 2 33 apache2 22230 < getcwd res=28 path=/var/
www/html/client/manage \n', ' 671972 10:55:37.380703935 2 33 apache2 22230 > getc
wd \n', ' 671949 10:55:37.380663961 2 33 apache2 22230 < stat res=0 path=/var/w
w/html/config/ourphp_language.php \n', ' 671948 10:55:37.380662287 2 33 apache2 2
2230 > stat \n', ' 671947 10:55:37.380660534 2 33 apache2 22230 < lstat res=0 pat
h=/var/www/html/client/manage/../../config/ourphp_language.php \n']"
```

FIGURE 4.17: First Syscall trace of the Alert for the Exploitation of cve-2023-30212

While both traces share few common syscalls, the results of the Explainer indicate that the attacks are very similar, as seen in Figures 4.18 and 4.19, with the same three most influential features highlighted.

4.2.2 Host Evaluation

Finally, to evaluate how the model handles a more complex and diverse scenario, it will be tested with the syscalls of the entire system. The scenario that will evaluate the model is a simple Nmap execution, where it should be detected no matter the circumstances, given the tool's capability in enumeration and discovery.

Firstly, to attest to the importance of anomaly detection, a signature will be created in Wazuh, where the agent will send syslogs of the system to the Wazuh instance, as seen in the listing 4.4.

```
<localfile>
  <log_format>syslog</log_format>
  <location>/var/log/syslog</location>
  <frequency>60</frequency>
</localfile>
```

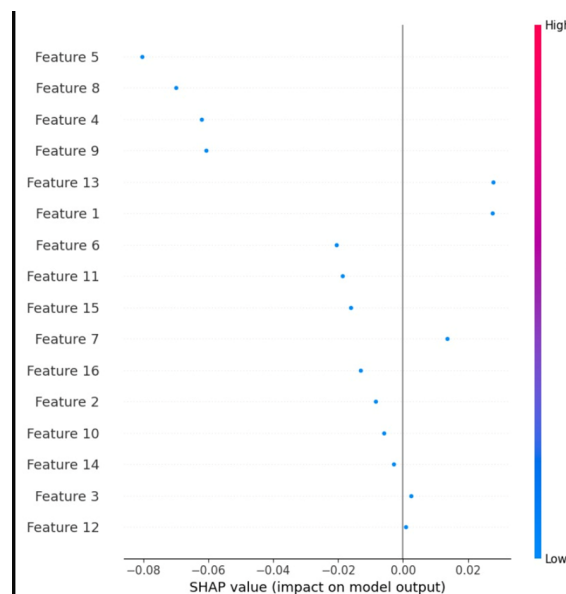


FIGURE 4.18: First explainer Output for the XSS of the Vulnerability cve-2023-30212.

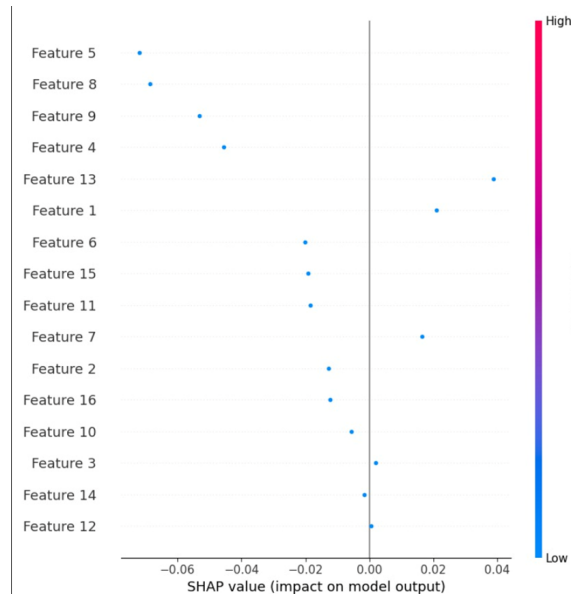


FIGURE 4.19: Second explainer Output for the XSS of the Vulnerability cve-2023-30212.

LISTING 4.4: Configuration of Wazuh Agent to send Syslogs

After the agent sends the syslogs, a simple rule in the Wazuh manager triggers an Alert when, in the syslogs, the execution path is the default of the Nmap usage, as seen in the listing 4.5. This simple rule is an example of a signature added to the Wazuh to detect a possible indicator.

```
<group name="nmap,syslog,command_monitoring">
  <rule id="100003" level="10">
    <description>nmap command detected in syslog</description>
    <match>exe="/usr/bin/nmap"</match>
    <group>nmap,</group>
  </rule>
</group>
```

LISTING 4.5: Wazuh Manager Rule to detect Nmap Execution

Figure 4.21 shows two different executions of the Nmap; the first is using the default method, where the installed binary is executed, and the second execution is using an already pre-compiled binary and, thus, not using the intended path of the program, using this simple method, as can be seen in figure 4.20, it avoids the signature created and successfully avoid detection.


```

\nSyscall Detected: [' 2457014 22:54:59.485021888 0 1000 nmap 10237 < connect res
=-115(EINPROGRESS) tuple=192.168.1.107:56098->192.168.1.9:80 fd=11(<4t>192.168.1.1
07:56098->192.168.1.9:80) \n', ' 2456851 22:54:59.484738210 0 1000 nmap 10237 > c
onnect fd=11(<4>) addr=192.168.1.9:80 \n', ' 2456848 22:54:59.484735635 0 1000 nm
ap 10237 < setsockopt res=0 fd=11(<4>) level=0(UNKNOWN) optname=0(UNKNOWN) val=...
. optlen=4 \n', ' 2456846 22:54:59.484733421 0 1000 nmap 10237 > setsockopt \n',
' 2456844 22:54:59.484732440 0 1000 nmap 10237 < setsockopt res=-1(EPERM) fd=11(<4
>) level=1(SOL_SOCKET) optname=25(SO_BINDTODEVICE) val=0 optlen=4 \n', ' 2456843
22:54:59.484731267 0 1000 nmap 10237 > setsockopt \n', ' 2456839 22:54:59.4847286
93 0 1000 nmap 10237 < setsockopt res=0 fd=11(<4>) level=1(SOL_SOCKET) optname=13
(SO_LINGER) val=..... optlen=8 \n', ' 2456836 22:54:59.484725608 0 1000 nmap 1
0237 > setsockopt \n', ' 2456834 22:54:59.484725077 0 1000 nmap 10237 < fcntl res
=0(<f>/dev/pts/1) fd=11(<4>) cmd=5(F_SETFL) \n', ' 2456832 22:54:59.484724236 0 1
000 nmap 10237 > fcntl fd=11(<4>) cmd=5(F_SETFL) \n', ' 2456831 22:54:59.48472367
4 0 1000 nmap 10237 < fcntl res=2(<f>/dev/pts/1) fd=11(<4>) cmd=4(F_GETFL) \n']",

```

FIGURE 4.22: Nmap syscall trace detected as an Anomaly by the Architecture

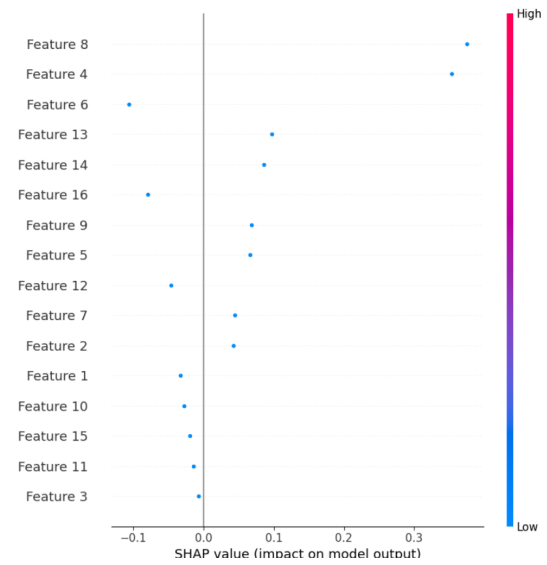


FIGURE 4.23: Explanation of the Nmap detection

4.3 Results

The architecture designed in chapter 3 entailed positive results, being able to timely detect attacks the model was not previously trained on and that did not have any prior knowledge attesting the capabilities of syscall as a source of great value in helping detect zero-day attacks.

The explainer also yielded positive results, providing insight into the potential triggering reasons for the alerts. Observing all the SHAP results for container attacks, similar vulnerabilities, CVE-2021-42013 and CVE-2022-33891, showed a very similar spread of the features that influenced detection. In contrast, other CVEs resulted in noticeably different SHAP patterns. This research concludes that XAI frameworks can be valuable for SOC analysts by offering detailed explanations of anomaly detections. When features are mapped to specific attacks, labeling each feature, or a combination of features, with techniques such as RCE, SSRF, or others, can highlight the factors influencing detection the most.

The table 4.2 summarizes the SHAP results, highlighting the most important feature for detection along with its value and other significant features. It is important to note that a comprehensive analysis of the SHAP values is better understood by examining the entire distribution; for example, the values of RCEs tend to be very similar in their distributions. The SHAP result for the host is not included in this table due to the variation

Vulnerability	Type	Most Impactful Feature	SHAP Value	Other Relevant Features
CVE-2021-42013	Remote Code Execution	Feature 8	0.05	Feature 16 Feature 4
CVE-2022-33891	Remote Code Execution	Feature 4	0.048	Feature 8 Feature 7
CVE-2022-42889	Remote Code Execution	Feature 7	0.04	Feature 14 Feature 4
CVE-2022-42889	Server Side Request Forgery	Feature 14	0.1	Feature 4 Feature 8
CVE-2023-7172	SQL Injection	Feature 14	0.13	Feature 13 Feature 8
CVE-2023-30212	Cross Site Scripting	Feature 13	0.03	Feature 1 Feature 7

TABLE 4.2: SHAP Results for the Containers Test

in the number of syscalls used for the detection, which causes the results not to be as reliable as the containers.

The results also show the limitations of the modern LID-DS dataset. Although the construction of the anomaly model entailed very low false positive results when used in a system with multiple behaviors, it did not perform well. The main reason for this is that the dataset is constructed using just syscall traces of containers, and containers, although very useful, are not capable of representing the complexity of a modern system, leaving the need for a publicly available dataset that can emulate the necessary behavior of modern hosts not only to have data mining results but to be used in a real-world cybersecurity context effectively. However, the dataset can be used to protect container infrastructure.

Chapter 5

Conclusion

Given the ever-evolving threat landscape, novel vulnerabilities, complex dependencies, and sophisticated exploits, detection systems that can determine malicious behavior without a signature and present a clear alert to the security professional are more needed than ever.

This thesis explores these needs using publicly available datasets, open-source tooling, and the latest machine-learning algorithms to explore how to approach these challenges and how effective the currently available techniques are.

5.1 Research Summary

The thesis explored how HIDS strategies change depending on the endpoint's Operating system, focusing on the two most used Operating systems, Linux and Windows, and how different data sources detect possible intrusion with each architecture.

It was reviewed how meaningful features are extracted from those data sources and what datasets researchers created that can leverage those data sources.

It also reviewed how machine learning is used to construct HIDS, leveraging concepts in training models that can predict and protect from previously unseen attacks, and how frameworks and tools, especially open-source, in this field help the security team better understand those detections.

The thesis explained the contemporary cybersecurity landscape, its inherent challenges, and various attacks. It also underscored the instrumental role of open-source projects in combating these threats and how businesses can harness these tools to shield themselves from modern-day risks.

Based on all of these concepts, an architecture was proposed that leveraged this knowledge to create a proof of work that these methods can be applied in a real-world scenario. The architecture leverages publicly available datasets and frameworks and a novel implementation of XAI in the context of interpretability of how deviations in the system behavior could be used in detecting unseen attacks.

With the architecture in place, the next step was to test it in real-world attacks and scenarios. The results were promising but also revealed some limitations, particularly due to the constraints of the dataset used. This underscored that while datasets and models can yield impressive results in the data science realm, they cannot be directly applied to real-world systems. Instead, they necessitate domain expertise and manual interventions to ensure practical applicability.

The results also showed that implementing the XAI framework, SHAP can positively impact the digestion and interpretability of anomaly alerts. This gives the analyst a degree of certainty that the anomaly model indicates malicious behavior. With further improvement of the solution, the alert can be mapped to a specific attack.

5.2 Current Limitations

Given that the thesis primarily focused on the detection and interpretability of intrusions in hosts rather than on creating a dedicated dataset, the available options were relatively limited. This became particularly evident when considering modern operating systems. Consequently, the dataset's quality significantly influenced the thesis results, especially when the architecture was expanded into a more complex environment, leading to increased False Positives.

Additionally, the development of the machine learning model for anomaly detection was constrained by hardware and time limitations, preventing further expansion of the model's complexity.

The testing environment was also restricted. The system was tested on only one machine with a limited set of behaviors, and containerized environments were not tested in production, which affected how an actual live environment might behave.

Moreover, the tests were limited to a few CVEs and a single host. To further consolidate this thesis's contributions, future work should employ a more comprehensive testing environment with a broader range of attacks to better evaluate the limitations of the XAI framework in interpreting anomaly alerts.

5.3 Future Work

Future work should explore the XAI frameworks in more depth in alert digestion so that analysts can better understand the alerts generated by machine learning tools. This should include improving testing, creating more scenarios with more complexity, and using techniques such as perturbation of data to validate even further the contributions of such frameworks to detection analysis. The alert should also map features and SHAP values to known techniques so that it can provide the analysis with more well-rounded information.

Also, the creation of a publicly available dataset with a modern operating system that recreates attacks and behavior of an entire system must be explored so that models can be created with more accurate data, not just using containers, finally creating datasets for Windows operating systems truly using the data source that represents the system behavior, exploring the soon to be released ebpf for Windows tooling, would be of great value given that it still the most used operating system and is the target of most of the attacks.

It's imperative to explore how current models and techniques affect system performance, memory consumption, and CPU usage. This research is important for understanding how to limit the usage so that the HIDS can be as transparent to the final user as possible. The best practices to harden and secure these systems against tampering attempts should also be explored.

5.4 Conclusions

This project explored the many aspects of the HIDS field, the primary data sources leveraged by modern solutions to detect indicators of compromise and known signatures, and the key concepts that make the solution viable in contemporary environments.

Appendix A

Appendix A

Python Library	Version
huepy	Any version
docker	4.3.1
numpy	1.19.4
pexpect	4.8.0
terminaltables	3.1.0
docopt	0.6.2
pandas	1.1.4
pymongo	3.11.1
tables	Any version
python-dateutil	2.8.1
matplotlib	3.3.3
PyMySQL	0.10.1
setuptools	50.3.2
Faker	4.17.0
requests	2.25.0
urllib3	1.26.2
CouchDB	1.2
pypcapkit	0.15.5
scapy	2.4.4
wget	3.2
filesplit	3.0.2

tqdm	4.62.3
nest-asyncio	1.5.1
gensim	4.1.2
torch	1.9.1
treelib	1.6.1
pyshark	0.4.3
networkx	2.6.3
pydot	1.4.2
shap	0.42.1
MiniSom	2.2.9
scikit-learn	1.0.2
pytest	7.1.2
tabulate	0.8.10
shap	0.42.1

TABLE A.1: Python Libraries Used in this Project

Appendix B

Appendix B

```
def filter_syscall(self, parsed_syscall):
    """
    return false if syscall is filtered out
    """
    print(parsed_syscall)
    if parsed_syscall[8] == "futex":
        return False
    if parsed_syscall[8] == "wait4":
        return False
    if parsed_syscall[8] == "times":
        return False
    if parsed_syscall[8] == "epoll_wait":
        return False
    if parsed_syscall[8] == "epoll_ctl":
        return False
    if parsed_syscall[8] == "getsockname":
        return False
    if parsed_syscall[8] == "getdents64":
        return False
    if parsed_syscall[8] == "fstat":
        return False
    if parsed_syscall[8] == "openat": #Only Containers Exception
        return False
    if parsed_syscall[5] == "python3.7":
        return False
    if parsed_syscall[5].startswith("systemd"):
        return False
    if parsed_syscall[5] == "iptables":
        return False
    if parsed_syscall[5].startswith("runc"):
        return False
    if parsed_syscall[8] == "newfstatat":
        return False
```

```
if parsed_syscall[8] == "pread":
    return False
if parsed_syscall[8] == "clock_gettime":
    return False
if parsed_syscall[5].startswith("docker"):
    return False
if parsed_syscall[8].startswith("rt_sig"):
    return False
if parsed_syscall[8].startswith("get"):
    return False
if parsed_syscall[8] == "sendmsg":
    return False
if parsed_syscall[8] == "sigaltstack":
    return False
if parsed_syscall[8] == "recvmsg":
    return False
if parsed_syscall[8] == "switch":
    return False
if parsed_syscall[8] == "mmap":
    return False
if parsed_syscall[8] == "epoll_pwait":
    return False
if parsed_syscall[8] == "munmap":
    return False
if parsed_syscall[8] == "nanosleep":
    return False
if parsed_syscall[8] == "mprotect":
    return False
if parsed_syscall[5].startswith("containerd"):
    return False
if parsed_syscall[5] == "sysdig":
    return False
if parsed_syscall[5] == "networkd-dispat":
    return False
if parsed_syscall[5] == "dbus-daemon":
    return False
if parsed_syscall[5].startswith("wazuh"):
    return False
if parsed_syscall[5] == "netstat":
    return False
if parsed_syscall[5] == "exe":
    return False
if parsed_syscall[5] == "networkctl":
    return False
if parsed_syscall[5] == "net-interface-h":
    return False
if parsed_syscall[5] == "df":
```

```
        return False
    if parsed_syscall[5] == "sed":
        return False
    if parsed_syscall[8] == "madvise":
        return False
    return True
```

LISTING B.1: All of the syscalls and users excluded in the Host evaluation

Bibliography

- [1] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*, ser. SECPRI-96. IEEE Comput. Soc. Press. [Online]. Available: <http://dx.doi.org/10.1109/SECPRI.1996.502675> [Cited on pages vii, ix, 7, 9, 10, 15, and 19.]
- [2] A. Nelson, *Incident Response Recommendations and Considerations for Cybersecurity Risk Management:: A CSF 2.0 Community Profile*. NIST, 2024. [Online]. Available: <http://dx.doi.org/10.6028/NIST.SP.800-61r3.ipd> [Cited on page 1.]
- [3] K. A. Scarfone and P. M. Mell, *Guide to Intrusion Detection and Prevention Systems (IDPS)*, 2007. [Online]. Available: <http://dx.doi.org/10.6028/NIST.SP.800-94> [Cited on page 1.]
- [4] J. R. Vacca, *Computer and Information Security Handbook*. Morgan Kaufmann Publishers, 2017. [Cited on page 5.]
- [5] G. Creech, "A framework for the evaluation of the theoretical threat coverage provided by intrusion detection systems," *2017 Military Communications and Information Systems Conference (MilCIS)*, Nov 2017. [Cited on page 5.]
- [6] Q. Huang, X. Xia, D. Lo, and G. C. Murphy, "Automating intention mining," *IEEE Transactions on Software Engineering*, vol. 46, no. 10, pp. 1098–1119, 2020. [Cited on pages 5 and 6.]
- [7] B. Subba and P. Gupta, "A tfidfvectorizer and singular value decomposition based host intrusion detection system framework for detecting anomalous system processes," *Computers and Security*, vol. 100, p. 102084, Jan. 2021. [Online]. Available: <http://dx.doi.org/10.1016/j.cose.2020.102084> [Cited on pages xiii and 6.]

- [8] W. Haider, "Developing reliable anomaly detection system for critical hosts: a proactive defense paradigm," Ph.D. dissertation, 2018. [Online]. Available: <http://hdl.handle.net/1959.4/61219> [Cited on pages 6, 7, 11, and 17.]
- [9] Y. Jiao and P. Du, "Performance measures in evaluating machine learning based bioinformatics predictors for classifications," *Quantitative Biology*, vol. 4, no. 4, p. 320330, Dec. 2016. [Online]. Available: <http://dx.doi.org/10.1007/s40484-016-0081-2> [Cited on page 7.]
- [10] Z. T. Sworna, Z. Mousavi, and M. A. Babar, "Nlp methods in host-based intrusion detection systems: A systematic review and future directions," *Journal of Network and Computer Applications*, vol. 220, p. 103761, Nov. 2023. [Online]. Available: <http://dx.doi.org/10.1016/j.jnca.2023.103761> [Cited on pages 7, 17, 19, and 20.]
- [11] G. Creech and J. Hu, "A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns," *IEEE Transactions on Computers*, vol. 63, no. 4, p. 807819, Apr. 2014. [Online]. Available: <http://dx.doi.org/10.1109/TC.2013.13> [Cited on pages 7, 12, and 16.]
- [12] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM Computing Surveys*, vol. 51, no. 5, p. 136, Nov. 2018. [Online]. Available: <http://dx.doi.org/10.1145/3214304> [Cited on pages 7, 12, 17, and 23.]
- [13] V. Radhakrishna, P. V. Kumar, and V. Janaki, "A temporal pattern mining based approach for intrusion detection using similarity measure," in *Proceedings of the The International Conference on Engineering ; MIS 2015*, ser. ICEMIS 15. ACM, Sep. 2015, p. 18. [Online]. Available: <http://dx.doi.org/10.1145/2832987.2833077> [Cited on page 7.]
- [14] G. Creech, "Developing a high-accuracy cross platform host-based intrusion detection system capable of reliably detecting zero-day attacks," Ph.D. dissertation, 2014. [Online]. Available: <http://hdl.handle.net/1959.4/53218> [Cited on pages 7, 9, 12, 13, 17, and 23.]

- [15] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and S. Gagnon, "A trace abstraction approach for host-based anomaly detection," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2015, pp. 1–8. [Cited on page 7.]
- [16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>. [Cited on pages 8, 19, 21, and 36.]
- [17] G. Srivastava, R. Jhaveri, S. Bhattacharya, S. Pandya, P. Rajeswari, and Reddy, G. Yenduri, J. Hall, M. Alazab, and T. Gadekallu, "Xai for cybersecurity: State of the art, challenges, open issues and future directions," 06 2022. [Cited on pages 8, 22, and 23.]
- [18] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the jam project," vol. 2, 02 2000, pp. 130 – 144 vol.2. [Cited on page 9.]
- [19] R. Lippmann, D. Fried, I. Graf, J. Haines, K. Kendall, D. McClung, D. Weber, S. Webster, D. Wyschogrod, R. Cunningham, and M. Zissman, "Evaluating intrusion detection systems: the 1998 darpa off-line intrusion detection evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2, 2000, pp. 12–26 vol.2. [Cited on pages 9 and 17.]
- [20] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 3, p. 186205, Aug. 2000. [Online]. Available: <https://doi.org/10.1145/357830.357849> [Cited on page 9.]
- [21] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," 01 2010, pp. 305–316. [Cited on page 9.]
- [22] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, ser. LISA '99. USA: USENIX Association, 1999, p. 229238. [Cited on page 9.]
- [23] V. Ravi, S. Kp, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," 09 2017, pp. 1222–1228. [Cited on page 9.]
- [24] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM conference on Computer and communications security*, ser.

- CCS02. ACM, Nov. 2002. [Online]. Available: <http://dx.doi.org/10.1145/586110.586145> [Cited on page 10.]
- [25] G. Borello, "Sysdig and Falco now powered by eBPF. — sysdig.com," <https://sysdig.com/blog/sysdig-and-falco-now-powered-by-ebpf/>, [Accessed 19-07-2024]. [Cited on pages xiii, 11, and 25.]
- [26] Y. Zhang, S. Luo, L. Pan, and H. Zhang, "Syscall-bsem: Behavioral semantics enhancement method of system call sequence for high accurate and robust host intrusion detection," *Future Generation Computer Systems*, vol. 125, p. 112126, Dec. 2021. [Online]. Available: <http://dx.doi.org/10.1016/j.future.2021.06.030> [Cited on pages xiii, 11, 23, and 37.]
- [27] S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture, "A host-based anomaly detection approach by representing system calls as states of kernel modules," 11 2013, pp. 431–440. [Cited on page 11.]
- [28] M. Liu, Z. Xue, X. He, and J. Chen, "Scads: A scalable approach using spark in cloud for host-based intrusion detection system with system calls," 09 2021. [Cited on page 12.]
- [29] M. Hutchins, "An Introduction to Bypassing User Mode EDR Hooks MalwareTech — malwaretech.com," <https://malwaretech.com/2023/12/an-introduction-to-bypassing-user-mode-edr-hooks.html>, [Accessed 19-07-2024]. [Cited on pages xiii and 13.]
- [30] F. A. Bin Hamid Ali and Y. Y. Len, "Development of host based intrusion detection system for log files," in *2011 IEEE Symposium on Business, Engineering and Industrial Applications (ISBEIA)*, 2011, pp. 281–285. [Cited on page 12.]
- [31] "Indicator Removal: Clear Windows Event Logs, Sub-technique T1070.001 - Enterprise | MITRE ATT&CK&xAE; — attack.mitre.org," <https://attack.mitre.org/techniques/T1070/001/>, [Accessed 03-03-2024]. [Cited on page 12.]
- [32] M. Topallar, M. Depren, E. Anarim, and K. Ciliz, "Host-based intrusion detection by monitoring windows registry accesses," in *Proceedings of the IEEE 12th Signal Processing and Communications Applications Conference, 2004.*, 2004, pp. 728–731. [Cited on page 12.]

- [33] “Modify Registry, Technique T1112 - Enterprise | MITRE ATT&CK” — [attack.mitre.org](https://attack.mitre.org/techniques/T1112/), <https://attack.mitre.org/techniques/T1112/>, [Accessed 03-03-2024]. [Cited on page 12.]
- [34] aviviano, “User Mode and Kernel Mode - Windows drivers — learn.microsoft.com,” <https://learn.microsoft.com/en-us/windows-hardware/drivers/gettingstarted/user-mode-and-kernel-mode>, [Accessed 19-07-2024]. [Cited on pages xiii and 14.]
- [35] CrowdStrike, “Technical Details: Falcon Update for Windows Hosts | CrowdStrike — crowdstrike.com,” <https://www.crowdstrike.com/blog/falcon-update-for-windows-hosts-technical-details/>, [Accessed 24-07-2024]. [Cited on page 14.]
- [36] “A Deep Dive Into Malicious Direct Syscall Detection - Palo Alto Networks Blog — paloaltonetworks.com,” <https://www.paloaltonetworks.com/blog/security-operations/a-deep-dive-into-malicious-direct-syscall-detection/>, [Accessed 24-07-2024]. [Cited on page 14.]
- [37] M. Xie, J. Hu, and J. Slay, “Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld,” in *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2014, pp. 978–982. [Cited on page 15.]
- [38] C. Ko, B. Fredericks, and R. Sekar, “A specification-based approach for building survivable systems,” in *Proceedings of the National Information Systems Security Conference*, 1998, pp. 288–297.
- [39] C. Warrender, S. Forrest, and B. Pearlmutter, “Detecting intrusions using system calls: alternative data models,” in *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*, 1999, pp. 133–145.
- [40] X. Hoang and J. Hu, “An efficient hidden markov model training scheme for anomaly intrusion detection of server applications based on system calls,” in *Proceedings. 2004 12th IEEE International Conference on Networks (ICON 2004) (IEEE Cat. No.04EX955)*, vol. 2, 2004, pp. 470–474 vol.2. [Cited on page 15.]
- [41] M. Xie, J. Hu, and J. Slay, “Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld,” *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, pp. 978–982, 2014.

- [Online]. Available: <https://api.semanticscholar.org/CorpusID:18583843> [Cited on page 15.]
- [42] W. Haider, J. Hu, Y. Xie, X. Yu, and Q. Wu, "Detecting anomalous behavior in cloud servers by nested-arc hidden semi-markov model with state summarization," *IEEE Transactions on Big Data*, vol. 5, no. 3, pp. 305–316, 2019. [Cited on page 15.]
- [43] A. Chaturvedi, S. Bhatkar, and R. Sekar, "Improving attack detection in host-based ids by learning properties of system call arguments," 01 2005. [Cited on page 15.]
- [44] F. Maggi, M. Matteucci, and S. Zanero, "Detecting intrusions through system call sequence and argument analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 381–395, 2010. [Cited on page 15.]
- [45] P.-F. Marteau, "Sequence covering for efficient host-based intrusion detection," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 994–1006, 2019. [Cited on page 16.]
- [46] S.-B. Cho and H.-J. Park, "Efficient anomaly detection by modeling privilege flows with hidden markov model," *Computers and Security*, vol. 22, pp. 45–55, 01 2003.
- [47] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and S. Gagnon, "A trace abstraction approach for host-based anomaly detection," in *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, 2015, pp. 1–8. [Cited on page 16.]
- [48] P.-F. Marteau, "Sequence covering for efficient host-based intrusion detection," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 994–1006, 2019. [Cited on page 16.]
- [49] S. Wunderlich, M. Ring, D. Landes, and A. Hotho, "The impact of different system call representations on intrusion detection," *Logic Journal of the IGPL*, vol. 30, 10 2020. [Cited on page 16.]
- [50] P. K. Mvula, P. Branco, G.-V. Jourdan, and H. L. Viktor, "Evaluating word embedding feature extraction techniques for host-based intrusion detection systems," *Discover Data*, vol. 1, no. 1, Mar. 2023. [Online]. Available: <http://dx.doi.org/10.1007/s44248-023-00002-y> [Cited on pages 16 and 36.]

- [51] G. Creech and J. Hu, "Generation of a new ids test dataset: Time to retire the kdd collection," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, 2013, pp. 4487–4492. [Cited on page 17.]
- [52] N. Goranin and D. eponis, "Investigation of awsctd dataset applicability for malware type classification," 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:219684959> [Cited on page 18.]
- [53] M. Grimmer, M. M. Röhling, D. Kreußel, and S. Ganz, "A modern and sophisticated host based intrusion detection data set," 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:216077392> [Cited on pages xiii, 18, and 19.]
- [54] M. Grimmer, T. Kaelble, F. Nirsberger, E. Schulze, T. Rucks, J. Hoffmann, and E. Rahm, *Dataset Report: LID-DS 2021*. Springer Nature Switzerland, 2023, p. 6373. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-35190-7_6 [Cited on page 18.]
- [55] C. C. Aggarwal, *Data Mining: The Textbook*. Springer International Publishing, 2015. [Online]. Available: <http://dx.doi.org/10.1007/978-3-319-14142-8> [Cited on page 19.]
- [56] S. Mulay, P. Devale, and G. Garje, "Intrusion detection system using support vector machine and decision tree," *International Journal of Computer Applications*, vol. 3, 06 2010. [Cited on pages 19 and 20.]
- [57] S. Suratkar, F. Kazi, R. Gaikwad, A. Shete, R. Kabra, and S. Khirsagar, "Multi hidden markov models for improved anomaly detection using system call analysis," in *2019 IEEE Bombay Section Signature Conference (IBSSC)*, 2019, pp. 1–6. [Cited on pages 19 and 20.]
- [58] S. S. Murtaza, W. Khreich, A. Hamou-Lhadj, and M. Couture, "A host-based anomaly detection approach by representing system calls as states of kernel modules," in *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp. 431–440. [Cited on pages 19 and 20.]
- [59] H. Satilmi, S. Akleyek, and Z. Y. Tok, "A systematic literature review on host-based intrusion detection systems," *IEEE Access*, vol. 12, pp. 27 237–27 266, 2024. [Cited on page 19.]

- [60] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, Mar. 2000. [Online]. Available: <http://dx.doi.org/10.1017/CBO9780511801389> [Cited on page 20.]
- [61] G. Kim, H. Yi, J. Lee, Y. Paek, and S. Yoon, "Lstm-based system-call language modeling and robust ensemble method for designing host-based intrusion detection systems," 2016. [Online]. Available: <https://arxiv.org/abs/1611.01726> [Cited on pages 20 and 37.]
- [62] S. Shende, "Long short-term memory (lstm) deep learning method for intrusion detection in network security," *International Journal of Engineering Research and*, vol. V9, 07 2020. [Cited on page 21.]
- [63] G. Rjoub, J. Bentahar, O. A. Wahab, R. Mizouni, A. Song, R. Cohen, H. Otrouk, and A. Mourad, "A survey on explainable artificial intelligence for cybersecurity," *IEEE Transactions on Network and Service Management*, vol. 20, no. 4, pp. 5115–5140, 2023. [Cited on pages 22 and 23.]
- [64] E. Mosca, F. Szigeti, S. Tragianni, D. Gallagher, and G. Groh, "Shap-based explanation methods: a review for nlp interpretability," in *Proceedings of the 29th international conference on computational linguistics*, 2022, pp. 4593–4603. [Cited on page 22.]
- [65] J. Dieber and S. Kirrane, "Why model why? assessing the strengths and limitations of lime," *arXiv preprint arXiv:2012.00093*, 2020. [Cited on page 22.]
- [66] "Home — docs.metasploit.com," <https://docs.metasploit.com/>, [Accessed 24-07-2024]. [Cited on page 23.]
- [67] "What is an Advanced Persistent Threat (APT)? - CrowdStrike — crowdstrike.com," <https://www.crowdstrike.com/cybersecurity-101/advanced-persistent-threat-apt/>, [Accessed 24-07-2024]. [Cited on page 23.]
- [68] "SolarWinds Supply Chain Attack | Fortinet — fortinet.com," <https://www.fortinet.com/resources/cyberglossary/solarwinds-cyber-attack>, [Accessed 24-07-2024]. [Cited on page 23.]

- [69] 262588213843476, “xz-utils backdoor situation (CVE-2024-3094) — gist.github.com,” <https://gist.github.com/thesamesam/223949d5a074ebc3dce9ee78baad9e27>, [Accessed 24-07-2024]. [Cited on page 23.]
- [70] Wazuh, “Use cases - Getting started with Wazuh — Wazuh documentation — documentation.wazuh.com,” <https://documentation.wazuh.com/current/getting-started/use-cases/index.html>, [Accessed 19-07-2024]. [Cited on pages xiii, 25, and 26.]
- [71] “OSSEC Documentation &x2014; OSSEC — ossec.net,” <https://www.ossec.net/docs/>, [Accessed 19-07-2024]. [Cited on page 25.]
- [72] “Documentation — elastic.co,” <https://www.elastic.co/docs>, [Accessed 19-07-2024]. [Cited on page 25.]
- [73] “What Is a Security Operations Center (SOC)? | IBM — ibm.com,” <https://www.ibm.com/topics/security-operations-center>, [Accessed 11-08-2024]. [Cited on pages 27 and 28.]
- [74] “How to prevent data breaches in 2023 — techstartups.com,” <https://techstartups.com/2023/01/04/prevent-data-breaches-2023/>, [Accessed 23-07-2024]. [Cited on page 27.]
- [75] “GitHub - LID-DS/LID-DS: LID-DS is an intrusion detection data simulation framework. — github.com,” <https://github.com/LID-DS/LID-DS>, [Accessed 17-07-2024]. [Cited on page 30.]
- [76] “Demo/Backend at master — LID-DS/Demo — github.com,” <https://github.com/LID-DS/Demo/tree/master/Backend>, [Accessed 25-07-2024]. [Cited on page 41.]
- [77] “NVD - CVE-2021-41773 — nvd.nist.gov,” <https://nvd.nist.gov/vuln/detail/CVE-2021-41773>, [Accessed 06-08-2024]. [Cited on page 49.]
- [78] “GitHub - Zeop-CyberSec/apache_normalize_path: Metasploit-Framework modules (scanner and exploit) for the CVE-2021-41773 and CVE-2021-42013 (Path Traversal in Apache 2.4.49/2.4.50) — github.com,” https://github.com/Zeop-CyberSec/apache_normalize_path, [Accessed 06-08-2024]. [Cited on page 49.]

- [79] “Apache Spark Command Injection Vulnerability (CVE-2022-33891) &x2013; Qualys ThreatPROTECT — threatprotect.qualys.com,” <https://threatprotect.qualys.com/2023/03/08/apache-spark-command-injection-vulnerability-cve-2022-33891/>, [Accessed 06-08-2024]. [Cited on page 52.]
- [80] “GitHub - HuskyHacks/cve-2022-33891: Apache Spark Shell Command Injection Vulnerability — github.com,” <https://github.com/HuskyHacks/cve-2022-33891>, [Accessed 06-08-2024]. [Cited on page 52.]
- [81] “GitHub - sunnyvale-it/CVE-2022-42889-PoC: CVE-2022-42889 (a.k.a. Text4Shell) RCE Proof of Concept — github.com,” <https://github.com/sunnyvale-it/CVE-2022-42889-PoC>, [Accessed 29-09-2024]. [Cited on page 53.]
- [82] “GitHub - gustanini/CVE-2022-42889-Text4Shell-POC: This repository contains a Python script to automate the process of testing for a vulnerability known as Text4Shell, referenced under the CVE id: CVE-2022-42889. — github.com,” <https://github.com/gustanini/CVE-2022-42889-Text4Shell-POC>, [Accessed 29-09-2024]. [Cited on page 53.]
- [83] “GitHub - sharathc213/CVE-2023-7172 — github.com,” <https://github.com/sharathc213/CVE-2023-7172>, [Accessed 28-09-2024]. [Cited on page 55.]
- [84] “GitHub - VisDev23/Vulnerable-Docker-CVE-2023-30212-: This contains the necessary files and Docker to replicate A vulnerability in OURPHP that has a XSS Vulnerability (CVE-2023-30212) — github.com,” <https://github.com/VisDev23/Vulnerable-Docker--CVE-2023-30212->, [Accessed 28-09-2024]. [Cited on page 56.]