Towards Intelligent Intrusion Detection Systems for Cloud Computing

Mohammed J. Aljebreen

Follow this and additional works at: https://repository.fit.edu/etd

Part of the Computer Sciences Commons
Towards Intelligent Intrusion Detection Systems for Cloud Computing

by

Mohammed J Aljebreen

A dissertation
submitted to Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Doctorate of Philosophy
in
Computer Science

Melbourne, Florida
June, 2018
Towards Intelligent Intrusion Detection Systems for Cloud Computing

by Mohammed J Aljebreen

William Allen, Ph.D.
Associate Professor, and Program Chair for Cybersecurity and Data Science
Computer Science Department
Committee Chair

Susan Earles, Ph.D.
Associate Professor
Electrical and Computer Engineering Department
Outside Committee Member

Philip Bernhard, Ph.D.
Associate Professor
Computer Science Department
Committee Member

Marius Silaghi, Ph.D.
Associate Professor
Computer Science Department
Committee Member

Philip Bernhard, Ph.D.
Associate Professor and Department Head
Director of School of Computing
ABSTRACT

Title:
Towards Intelligent Intrusion Detection Systems for Cloud Computing

Author:
Mohammed J Aljebreen

Major Advisor:
William Allen, Ph.D.

Statistics presented in the background chapter show the tremendous number of security breaches that occurred in recent years. Furthermore, securing the new advanced technologies has become more challenging. Cloud computing and containers are among those emerging technologies that have introduced security challenges, and it is necessary that they be addressed. One of the main features of both cloud and container technologies is the sharing of resources of both hardware and software. The challenge becomes more complex when running containers in a cloud environment. Thus, advancing the security of container in the cloud environment is necessary, especially because of the lack of research in this area.

This dissertation advances the current technologies for securing containers running in the cloud by proposing and implementing intelligent intrusion detection systems. Also, a data representation mechanism is proposed to speed up the process of modeling the classifiers and forming the samples. The results of the proposed systems meet the requirements of designing cloud-based intrusion detection systems and maintaining high True Positive Rates and low False Positive Rates.
# Table of Contents

Abstract iii

List of Figures viii

List of Tables ix

Acknowledgments x

1 Introduction 1

1.1 Overview ........................................ 1

1.2 Motivation ....................................... 4

1.3 Research Questions ............................... 5

1.4 Thesis Statement ................................. 6

1.5 Approaches and Contributions ................... 6

1.6 Dissertation Structure ............................ 7

2 Background 9

2.1 Intrusion Detection Systems (IDS) ................ 9

2.1.1 Intrusion Detection System Characteristics ...... 11

2.2 Cloud Computing ................................. 12

2.2.1 Essential Characteristics ....................... 13
2.2.2 Service Models ............................................. 14
2.2.3 Deployment Models ....................................... 14
2.2.4 OpenStack .................................................. 15
2.3 Containers ..................................................... 16
2.3.1 Cloud Intrusion Detection Systems (CIDS) ............... 17
2.4 Intrusion Detection System Taxonomy .......................... 19
2.4.1 Detection Methods .......................................... 20
2.4.2 Data source .................................................. 21
2.4.2.1 System Calls .............................................. 21
2.5 Data Processing Techniques .................................... 23
2.5.1 Data Labels .................................................. 24
2.5.2 Intrusion Detection Algorithms Categories ................ 25
2.6 Intrusion Detection Algorithms ................................ 25
2.6.1 Outlier detection ............................................. 26
2.6.2 Machine learning algorithms ................................. 29
2.6.3 Decision Tree C4.5 ........................................... 31
2.6.4 Random Forests .............................................. 33
2.6.5 Support Vector Machines (SVM) ............................ 34
2.7 Attacks on cloud computing .................................... 36

3 Literature Review ................................................. 38
3.1 Related Work ................................................... 47

4 Enriched Data Representation ....................................... 51
4.1 Data Source ..................................................... 52
4.2 Data Collection .................................................. 53
4.3 Data Pre-processing ........................................ 54
4.4 Sliding Window Technique ............................... 55
4.5 Bag of system calls representation .................... 56
   4.5.1 Issues with the Previous Representation ........... 57
4.6 Enriched Data Representation ........................... 60
   4.6.1 Constructing the Feature Vector ................... 61
   4.6.2 Building the Normal Profile ....................... 62
   4.6.3 Forming the Samples and the Attacks Patterns .... 63

5 Environment and Framework Designs 66
   5.1 The Environment Design ............................... 66
   5.2 Data Generation ....................................... 70
       5.2.1 Normal data ..................................... 70
       5.2.2 Abnormal data ................................. 71
   5.3 The Framework Design .................................. 72

6 Implementations and Results 76
   6.1 Type of Experiments ................................... 77
   6.2 Classifiers Training ................................... 78
   6.3 Cross-validation ...................................... 79
   6.4 Classifiers Evaluation ................................. 80
   6.5 Anomaly Intrusion Detection Systems ................ 82
       6.5.1 Interquartile range (IQR) Classifier ............. 82
           6.5.1.1 Classifier Training ......................... 82
       6.5.2 \( K \)-nearest Neighbor Classifier ................. 83
           6.5.2.1 Classifier Training ......................... 84
List of Figures

2.1 Intrusion Detection System Taxonomy . . . . . . . . . . . . . . . . . . 19
2.2 Anomalies in a 2-dimensional dataset [48] . . . . . . . . . . . . . . . 20
2.3 Box plot of a normal distribution N(0,2) [11] . . . . . . . . . . . . . 27
2.4 A multilayer perceptron with two hidden layers [66] . . . . . . . . . . 30
2.5 An example of a decision tree [27] . . . . . . . . . . . . . . . . . . . 32
2.7 Support vector machines illustration [16] . . . . . . . . . . . . . . . . 35
3.1 An Intrusion Severity System . . . . . . . . . . . . . . . . . . . . . . . 42
4.1 Strace sample . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
4.2 A sliding window size of 10 . . . . . . . . . . . . . . . . . . . . . . . 56
5.1 OpenStack Components [12] . . . . . . . . . . . . . . . . . . . . . . . . 67
5.2 OpenStack Architecture [12] . . . . . . . . . . . . . . . . . . . . . . . 68
5.3 OpenStack-Docker interactions[7] . . . . . . . . . . . . . . . . . . . . 69
5.4 The Proposed Intrusion Detection System Architecture . . . . . . . . 73
6.1 10-fold Cross Validation [45] . . . . . . . . . . . . . . . . . . . . . . . . 80
6.2 Index-Based Algorithm for Computing Anomalies . . . . . . . . . . . 84
6.3 Computation of Distance for 10th Nearest Neighbor . . . . . . . . . . 85
List of Tables

3.1 Security Requirements for a Virtual Machine .......................... 43

4.1 Threat Level Classification .............................................. 60

5.1 The main cloud components IP addresses ............................. 70

6.1 Confusion matrix for a two-class problem ............................ 80
6.2 Interquartile Range (IQR) Classifier Evaluation ..................... 83
6.3 K-nearest Neighbor Classifier Evaluation ............................ 86
6.4 Artificial Neural Networks (ANNs) Classifier Evaluation .......... 89
6.5 C4.5 Classifier Evaluation ........................................... 93
6.6 Random Forests Classifier Evaluation ................................ 95
6.7 SVM Classifier Evaluation ............................................ 99
6.8 C4.5 Classifier Evaluation with/without the distinct Feature ...... 101
6.9 Classifiers Evaluation when Dimensional Reduction is Applied ... 102
6.10 Classifiers Evaluation without Applying Dimensional Reduction .. 103
Acknowledgements

I would like to take this opportunity to acknowledge the many people who made this dissertation possible.

I would like to express my sincere gratitude to my mentor and advisor, Dr. William Allen. I have been fortunate to receive the benefit of his knowledge and experience over the past five years. I could not have asked for a better scholarly role model than Dr. Allen, and I am truly thankful and grateful for his guidance and involvement throughout my studies and research. I would also like to sincerely thank my committee members, Dr. Philip Bernhard, Dr. Marius Silaghi, and Dr. Susan Earles, for providing valuable comments on my dissertation.

I hold very special appreciation for Dr. Marco Carvalho and Dr. Heather Crawford for dedicating part of their labs at the Harris Institute for Assured Information and the IoT & Hardware Security labs to the implementation of my research experiments. My sincere thanks also goes to Dr. Mehadi Hassen, Dr. Thomas Eskridge, and Dr. Phil Chan, for their insights and suggestions concerning my research.

I am indebted to my mother, Monira, for her endless love and patience.
I am also indebted to my loving father, Jebreen, who passed away with dreams that I would one day earn my doctorate degree. I am also grateful to my wife, Amjad, for her encouragement and patience while I was busy with research. I am thankful to my sisters and brothers for their affection and encouragement. My profound gratitude to my lovely son, Yasser, who is too young to realize that his soul and love have been ongoing inspiration. I would also like to thank my friends, Dr. Abdullah Alharbi and Dr. Moteeb Almoteri for their encouragement.

Last but not the least, I am grateful to my country, Saudi Arabia, represented by the King Saud University; without their generous financial support and encouragement, this research would not have been possible.
Dedicated to Yasser
Chapter 1

Introduction

1.1 Overview

The evolution of advanced computing technologies is expanding rapidly, introducing various new business and operating models. One of the emerging technologies that has become popular in the past decade is cloud computing.\(^1\) One of its main features and benefits is shared network resources whereby multiple clients share the same hardware using logical isolation technologies. This feature ensures scalability, flexibility, and availability through on-demand provisioning of those resources.

One of the new technologies that was introduced in the past few years and which can be integrated into the cloud, is *containers*. The container is “a

\(^1\)Chapter 2 introduces the background and more detail about the cloud technologies and its characteristics.
lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings” [9]. A container can run on the same instance and share the same OS kernel with other containers whereby each one runs as isolated processes in user space [9].

With all the above mentioned benefits of the shared network resources in the cloud and the sharing of kernels between containers, many cloud solutions do not provide the necessary security among clients [3]. Also, there are many security risks related to containers and the main ones are listed as follows [14]:

- Unsecured communication and unrestricted network traffic.
- Unrestricted access of process and files
- Kernel level threats

The above risks for sharing resources in a cloud environment between clients and sharing the kernel among containers could introduce risks that affect security goals. In the information security field, the CIA triad is a well-known model which consists of three main core goals of security: confidentiality, integrity, and availability. Since this CIA triad does not cover new threats as mentioned in [49], Cherdantseva and Hilton identified additional goals that include: accountability, auditability, authenticity/trustworthiness, non-repudiation and privacy. The International Standard (ISO27000) [22] defines the goals (confidentiality, non-repudiation) as follows:

- Confidentiality: The property that information is not made available or dis-
closed to unauthorized individuals, entities, or processes.

• **Integrity**: The property of accuracy and completeness.

• **Availability**: The property of being accessible and usable upon demand by an authorized entity.

• **Authenticity/Trustworthiness**: The property that an entity is what it claims to be.

• **Non-repudiation**: The ability to prove the occurrence of a claimed event or action and its originating entities.

• **Auditability**: Cherdantseva and Hilton defined auditability as an ability of a system to conduct persistent, non-bypassable monitoring of all actions performed by humans or machines within the system.

• **Privacy**: Cherdantseva and Hilton also defined privacy as a system which should obey privacy legislation and should enable individuals to control, where feasible, their personal information (user-involvement).

Having briefly discussed the goals of information security, in the real world, organizations and customers are still exposed to a tremendous number of breaches and attacks. There are infinite numbers of breach sources, such as hacking and employee-driven factors (i.e. errors and negligence), but this topic is outside the scope of this research and is not discussed.

In 2017, the Identity Theft Resource Center (ITRC) published statistics showing that more than 171 million records were exposed during the year 2017
solely [6]. Those exposure affected records that fall into the following categories: banking/credit/financial, business, educational, government/military, and medical/health care [6]. Those attacks and breaches could make a huge impact on the customers or on the organization as a whole; however, Pfleeger in his book [134] stated that dealing with harm is sought in several ways as follows:

- Prevent it, by preventing attackers from doing what they intended to do or closing vulnerabilities.
- Deter it, by mitigating attacks
- Deflect it, by deluding attackers to attack other attractive targets
- Detect it as it happens or sometimes after the fact
- Recover from their effects

Each one of these tasks has its own procedures in order to design and implement a system with the above capabilities. This research is focused primarily on how to detect these breaches and attacks.

1.2 Motivation

Since the cloud environment have the network resources shared among consumers, the risk becomes higher, especially when running containers in the cloud environment since the containers also share the kernel between containers by nature.
Protecting the containers running in the cloud environment is necessary since their features and benefits introduce risks that must be prevented or at least mitigated. As discussed earlier, detecting the potential intrusions is one of the ways can be done to identify risks against a system. A search of recent publications found no research that concerning the development of an intrusion detection system for containers running in the cloud environment. Thus, this research develops an intrusion detection system that is suitable for containers and cloud technologies.

1.3 Research Questions

Having discussed the potential risks of having containers run in the cloud environment, many research questions are investigated and answered throughout the dissertation. The questions are as follows:

- Is it possible to build a host-based intrusion detection system for containers running in the cloud environment?

- Is it possible to use system calls as a data resource of a container to build a normal behaviour that represents that container?

- How can the container data effectively be represented in order to build classifiers very quickly and efficiently?

- How can unknown attacks against containers be detected using machine learning techniques?
• How can a cloud-based intrusion detection system be designed to maintain high detection rates with low overhead.

1.4 Thesis Statement

*It is possible to build an intrusion detection system for containers running in the cloud environment. The presented enriched data representation and framework allows the development of accurate, efficient, and intelligent intrusion detection systems for cloud computing using machine learning algorithms.*

1.5 Approaches and Contributions

In this research, an intrusion detection system was developed for containers running in the cloud environment. As described earlier, a review of recent research shows that there are no existing intrusion systems for cloud-based containers. Thus, this reason motivated the authors to proceed into this research. There are several requirements for the development of a cloud-based intrusion detection system which are listed in Chapter 2 and which this research meets.

One of the main characteristics of a cloud-based intrusion detection system, is that it works in real-time and detects a variety of attacks with minimal False Positive Rates. Thus, in this research a data representation mechanism was developed to ensure that an intrusion detection system could learn the normal model of a container very quickly and accurately which helps to achieve detection
with maximum true positive rate and minimum false positive rate.

The main contributions of this research are as follows:

- Developing an intrusion detection system for containers running in the cloud environment. The literature chapter shows no existing published research that has accomplished this technological advancement.

- Designing and implementing a cloud environment that was used to collect real world data that was used for test the proposed intrusion detection system.

- Developing a data representation mechanism that allows the intrusion detection systems classifiers to learn the model quickly and accurately.

- Designing and developing an intrusion detection system framework for both anomaly and misuse detection techniques.

- Applying and implementing sophisticated machine learning algorithms and evaluating their performance and detection abilities.

1.6 Dissertation Structure

This dissertation is organized as follows: Chapter 2 describes the backgrounds of the emerging technologies used in this research. Also, it provides a brief history and descriptions of the intrusion detection techniques that were developed. In addition, it describes the attacks that threaten the cloud and the container environments. Chapter 3 reviews the literature of the cloud-based intrusion detection systems. It also discusses the current related research relative to the proposed systems in this
dissertation and how they differ. Chapter 4 describes, in detail, the Enriched Data Representation of the data source. It also explains how the data were generated and collected as well as their pre-processing and construction. Chapter 5 shows the designs of the environment used and the proposed intrusion detection framework. Chapter 6 describes the intrusion detection implementations and how the classifiers were trained and tested to detect intrusions. It also provide the results of each technique as well as providing a discussion showing a comparison the different models. Chapter 7 summarizes the development of the proposed systems and their results; as well as listing the limitations and suggestions for future work.
Chapter 2

Background

2.1 Intrusion Detection Systems (IDS)

This research is focused on the detection aspect that was discussed in the Introduction. To detect potential risk against a system, the system needs to be monitored and capable of detecting an intrusion. What is Intrusion Detection? Heady et al. [75] defined intrusion as “any set of actions that attempt to compromise the integrity, confidentially, or availability of a resource”. Stephen Smaha in [149] classified the intrusions into the following types:

- **Attempted break-ins** by unauthorized users which can be detected by atypical behavior profiles or violations of security constraints.

- **Masquerade attacks** which occur when intruders try to convince the system that they are authorized to access. It also can be detected by atypical be-
haviour profiles or violations of security constraints.

- **Penetration of the security control system** which happens when an intruder attempts to modify system characteristics. It can be detected by using privileged logins.

- **Leakage**, that causes moving information out of the system. It can be detected by atypical usage of I/O resources.

- **Denial of service** that makes resources unavailable for legitimate users. It can be detected by atypical usage of the resources.

- **Malicious use**, such as deleting files and exhausting resources can also be detected by atypical behavior profiles, violations of security constraints, or usage of special privileges.

Decades ago, system administrators monitored the systems needed to personally watch the activities’ monitor and look for unusual activities [88]. An improvement to this procedure occurred in the late ’70s when administrators audited logs by printing the logs’ files and searching through them. It was time consuming and very easy to miss the suspicious logs. After storage became less expensive, the logs moved online and programs were developed to analyze the data. However, the analysis process was slow and computationally intensive [88].

A real-time intrusion detection system was developed in the early ’90s whereby audit data is reviewed as it is produced [88]. Since then, the developed intrusion detection systems’ effectiveness was based on how fast the system could detect intrusions, maximizing the True Positive rate and minimizing the False
Positive rate [36] as described in detail in Chapter 6.

### 2.1.1 Intrusion Detection System Characteristics

There are several characteristics that are needed to develop an IDS in order to have optimized performance, maximizing the detection rate and minimizing the errors. Patel et al. in [144]; Sharma and Sinha in [130] listed numerous characteristics that should be in IDS. Patel et al. in [131] summarized those characteristics as follows:

1. Runs continuously without human supervision.

2. Is fault tolerant to able to recover from crashes.

3. Is simply tailored to a specific network.

4. Adapts to behaviour changes of user/system over time.

5. Works in real-time.

6. Detects maximum number of intrusions with minimum number of false-positive alarms.


8. Is self-configurable to the security policies changes.

9. Operates while maintaining minimum overhead
With the advanced technologies that involve sharing data and resources, the IDS must keep up with the amount of data and number of involved networking components and devices that process the data. As stated in the Introduction, cloud computing technology is one of the popular emerging technologies that involve a tremendous number of resources that work together and process huge amounts of data. The intrusion detection systems in the cloud environment require certain characteristics to keep up with those introduced challenges. Before going into detail about those requirements, a brief description of cloud computing technologies and its characteristics and services will show the need to develop an intrusion detection system that can be applied to these advanced technologies.

2.2 Cloud Computing

The National Institute of Standards and Technology (NIST) defined cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”[2] According to NIST, the cloud model is composed of five essential characteristics, three service models, and four deployment models. [133]
2.2.1 Essential Characteristics

- **On-demand self-service**: A cloud consumer can be provided for automatically by computing resources, with no need to interact with a cloud provider.

- **Broad network access**: Resources are available over the network and can be accessed using different platforms, such as laptops, mobiles, etc.

- **Resource pooling**: The resources are pooled to serve consumers in a multi-tenant model.

- **Rapid elasticity**: The resources can be elastically provisioned and sometimes automatically in case of scaling.

- **Measured service**: The cloud resources can be monitored and controlled transparently for the CP and the consumer.

Additional characteristics listed in [131] that should be considered when designing a cloud-based intrusion detection system are as follows:

- **Reliability**, which is the capability of the system to continue its operation without disruption.

- **Quality of Service (QoS)** which ensures the importance of the support of specific requirements that should be met through the provided services or resources. This can be met by ensuring that the agreed service quality of the cloud user is in the Service Level Agreement (SLA) which covers response time, throughput, safety, etc.
• **Agility and adaptability** refer to on-time reaction to the size changes of resources and the amount of those requests where they should be adapted automatically by the resources management.

• **Availability** is the ability to provide redundant services and data to avoid service provision failures.

### 2.2.2 Service Models

• **Cloud Software as a Service (SaaS):** The capability of a consumer to use the provider’s applications running on a cloud infrastructure. The cloud provider responsibly manages the underlying cloud infrastructure.

• **Cloud Platform as a Service (PaaS):** The capability of a consumer deploying acquired applications onto the cloud infrastructure using tools supported by the provider.

• **Cloud Infrastructure as a Service (IaaS):** The capability of a consumer to get computing resources (e.g. storage, network) that can deploy, which includes any software, whether operating system or application.

### 2.2.3 Deployment Models

• **Private cloud:** In this model, the cloud infrastructure provided is owned, operated, and managed by an organization or by a third party or both.

• **Community cloud:** The cloud infrastructures in this model are shared between certain organizations, which support a certain community with shared
concerns.

- **Public cloud**: The cloud infrastructure provided is available to the public and is owned by a certain organization that sells services.

- **Hybrid cloud**: The cloud infrastructure is composed of two or more private, community, or public clouds.

### 2.2.4 OpenStack

OpenStack is "a cloud operating system that controls large pools of computing, storage and networking resources throughout a datacenter, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface." [12]

Choosing what cloud platform to use for this research experiments was a challenge. OpenStack was the preferred option for several reasons.

- Free open source that has a huge community that contributes to enhancing its features.

- Trusted platform that has been used and supported by the leading companies such as AT&T, Walmart, RedHat, Canonical, Dell, IBM, HP, Cisco, and PayPal.

- Flexibility that can be integrated with other services and technologies.

- Compatibility; OpenStacks APIs are designed to be compatible with public cloud platforms [23].
• Security; high level of security can be achieved through role-based access controls [23].

Chapter 5 shows in detail the components of the OpenStack environment and how they are configured.

2.3 Containers

Containers are defined as “a method of operating system virtualization that allows you to run an application and its dependencies in resource-isolated processes.” [19]

Compared to virtual machines, containers have similar resource isolation and allocation benefits. However, they function differently because containers virtualize the operating system instead of the hardware. Also, they are more portable and efficient [4].

As discussed in the Introduction, since the containers share the same kernel with other containers, the risk becomes high even with the isolation function especially when they run in the cloud environment because also, in the cloud, the resources are shared among users as well. Thus, the need to develop intrusion detection systems that detect any suspicious activities on containers is critical. A search of recent publications found no research about the development of intrusion detection systems for containers run in the cloud environment. That motivated this research to proceed as discussed in the introduction chapter.
2.3.1 Cloud Intrusion Detection Systems (CIDS)

Developing a cloud-based intrusion detection system requires additional requirements due to its complex architecture and integrated services that interact together as well as with outside services. In terms of performance, there are several research papers that evaluated the applications’ performance running in cloud environment and in traditional servers. For instance, Jackson et al. [82] stated in their research that “EC2\(^1\) is six times slower than a typical mid-range Linux cluster, and twenty times slower than a modern HPC\(^2\) system. The interconnect on the EC2 cloud platform severely limits performance and causes significant variability.” Shea et al.[145] found that there are significant degradations and variations for VM communications with both TCP and UDP traffic. Wang et al.[163] measured the processor sharing, packet delay, TCP/UDP throughput, and packet loss among Amazon EC2 virtual machines and found that virtualization causes significant throughput instability and abnormal delay variations. For the cloud databases’ performance, Donkena Kaushik and Subbarayudu Gannamani in [61] found that cloud databases have poor performance in terms of response time while retrieving the data compared to the traditional databases.

Considering these effects on performance, the previously discussed traditional IDS requirements, and the characteristics of cloud computing, the minimum requirements for a cloud based intrusion detection system that was introduced in [131] are listed below:

---

\(^1\)Amazon Elastic Compute Cloud (EC2)

\(^2\)High-performance computing
• Handle large-scale, dynamic multi-tiered autonomous computing and data processing environments

• Detect a variety of attacks with minimal False Positive Rates

• Detect intrusions as they occur

• Self-Adaptive Autonomically, where the CIDS should adapt to changes in configurations as the computing resources are dynamically added and removed.

• CIDS Scalability which can handle a large number of network nodes.

• Be deterministic, where the CIDS should maintain the acceptable service-level agreement (SLA), be reliable, have high uptime service as well as maintaining minimum overhead.

• Synchronize the autonomous CIDS, whereby the collaborated intrusion detection systems must be synchronized in order to detect attacks in real-time.

The authors analyzed 10 cloud based intrusion detection/prevention systems ([157] [106] [160] [58] [127] [150] [90] [60] [154] and [83]) against those requirements and found that none of them met all the requirements; however, they met them partially or they were not applicable. As a result, there is an essential need to design a cloud base that meets the proposed requirements if they are applicable to the designed cloud environment.
2.4 Intrusion Detection System Taxonomy

The intrusion detection system can be classified into various categories, such as detection methods (misuse/anomaly), data source (host-based/network-based), analysis timing (real-time/offline), system structure (centralized/distributed), behavior after detection (passive/active) [59] and [88]. Letou et al.[110] has classified the intrusion detection systems into categories based on type of intruders (external/internal) and type of intrusions (leakage/malicious use/etc.).

In the following sections, the detection methods and the data source categories are explained in detail, as they are the focus of this research as shown in Figure 2.1.

Figure 2.1: Intrusion Detection System Taxonomy
2.4.1 Detection Methods

The IDS in general (this also applies to the cloud based IDS) is categorized based on its detection methods into two popular categories, *misuse detection* and *anomaly detection*. In misuse intrusion detection, (also called signature based detection) known patterns of intrusion (intrusion signatures) are used to identify intrusions as they happen [107], [102], [103], and [65].

Chandola in [48] defined anomalies as the “patterns in data that do not conform to a well defined notion of normal behavior.” The data in Figure 2.2 shows two normal regions, $N1$ and $N2$, and most of the observed data points lie in those two regions. The data points that are far from those regions, namely point $o1$ and $o2$ and points in region $O3$, are considered anomalies.

![Figure 2.2: Anomalies in a 2-dimensional dataset [48]](image-url)
In anomaly intrusion detection, it is assumed that the nature of the intrusion is unknown, but that the intrusion will result in behavior different from that normally seen in the system.

2.4.2 Data source

The IDS is categorized based on its data source into two main categories: network-based intrusion detection system (NIDS) and host-based intrusion detection system (HIDS). They are defined as that “HIDS provides protection for the host on which it is being installed whereas NIDS suspects for attacks or irregular behavior by inspecting the contents and header information of all the packets moving across the networks” [30] and [29].

In this research, the proposed intrusion detection system is a host-based system. Often in the host-based intrusion detection systems, system calls are the preferred source of data to characterize the normal behavior of applications [54].

In the next section, a brief explanation and history about the system calls is given, since they are also the source of data used in this research.

2.4.2.1 System Calls

The system call is defined as “the fundamental interface between an application and the Linux kernel.” [18]

Using system calls as the data source to build an IDS is now acknowledged to perform better with complex IDS detection engines than using different data
sources because system calls interact with the low-level operating system space and are able to access core kernel functions[54].

Forrest et al. in [65] are the first researchers who introduced developing an intrusion detection system using system calls as the data source. They used lookahead pairs technique where they recorded the number of mismatches as a percentage of the total possible number of mismatches which is called time-delay embedding (Tide). Their work consists of two stages: building a normal database that represents the normal behavior of the system, and comparing the new sequences to the normal database and finding the mismatches. The authors concluded that this could be implemented efficiently and would provide good results.

An improved system called sequence time-delay embedding (stide) was developed by the same team [77] and it worked better. They used the Hamming distance technique to find the differences between the tested sequences and the normal sequences. Assuming two sequences $i$, and $j$, the Hamming distance was denoted by $d(i,j)$ between the sequences. The minimal value of the Hamming distance, $d_{\text{min}}$, between the new sequences and the normal sequences was calculated by:

$$d_{\text{min}}(i) = \min \{d(i, j) \text{for all normal sequences } j\}$$

where $d_{\text{min}}$ represents the strength of the anomalous signal.

One of the methods that was used to develop this intrusion detection system was the frequency-based method which models the frequency distributions of various events. Damashek [57] developed a frequency-based method using an
n-gram vector to classify text documents. The documents are represented by a vector that is a histogram of sequence frequencies. Each element corresponds to one sequence of length $n$ called an $n$-gram and the value of the element is the frequency with which the n-gram occurs in the document. This method can be used to classify system calls to develop intrusion detection systems. However, this approach is not suitable for online testing because trace vectors cannot be evaluated while the program is running, and it has to be terminated [164]. Thus, it is not suitable to develop a cloud-based HIDS using this approach because the cloud IDS has to work in real-time, which is one of its requirements.

William Cohen in [51] developed RIPPER (Repeated Incremental Pruning to Produce Error Reduction), which is a rule learning system. RIPPER was used in [107] to investigate if a machine learning approach can be used to learn the normal and abnormal patterns. Their results showed that “machine learning can play an important role by generalizing stored sequence information to perhaps provide broader intrusion detection services.” [107] In this research, machine learning algorithms were also tested if they can learn intrusions in containers running in the cloud environment as discussed in Chapters 4 and 6.

2.5 Data Processing Techniques

After collecting the data, the intrusion detection technique uses that data to train its classifier in order to distinguish between normal and abnormal behaviors. The intrusion detection algorithm’s purpose is to report intrusions. The output of the detection techniques is categorized into two main types [48]:

23
- **Scores:** This is mainly used for anomaly detection techniques where an anomaly score is assigned to each tested instance that determines whether it is an anomaly or not, based on a predefined threshold.

- **Labels:** The detection techniques assign a label (e.g. normal or abnormal) for the tested instances. Determining whether the label is normal or abnormal depends on the type of algorithm technique type. The data labels are explained in detail in the next section.

### 2.5.1 Data Labels

The data samples are associated with labels that denote each sample as normal or abnormal. Labeling the normal behavior is easier than labeling the abnormal one because the abnormal is often dynamic in nature so new types of anomalies could have no labels in the training data [48]. Based on labels’ availability, the detection techniques operate in those modes:

- **Supervised detection**, where the detection algorithm is trained using labels for normal and abnormal samples.

- **Semi-supervised detection**, where only normal samples are labelled and the abnormal are not when training the detection algorithm.

- **Unsupervised detection** where the samples are not required to be labelled when the detection algorithm is trained.
2.5.2 Intrusion Detection Algorithms Categories

For each mode of the detection techniques mentioned in section 2.5.1, there are different detection algorithms that fall under it. The detection algorithms are categorized into different models. The main ones that have been used in this research are briefly explained as follows:

- **Statistical based:** These are used by anomaly detection techniques that use statistical theories to assign anomalies’ scores.

- **Instance based:** This is also called the lazy-learning algorithm [122] because it delays the generalization or induction processes until the classification process ends. $K$-nearest neighbor is the most popular instance-based algorithm which is based on the principle that dataset instances generally exist in close proximity to those instances with similar properties [53] [98].

- **Classification based:** This uses the training data to model a classifier that assigns intrusion scores or labels to a tested instance [48].

The next section briefly describes the detection algorithms used in this research for the anomaly and misuse detection.

2.6 Intrusion Detection Algorithms

In this section, a brief description is given for different anomaly detection and machine learning classification techniques for intrusion detection systems that were
used in this research. The first section is for anomaly detection where some outlier algorithms were used to detect anomalies (outliers). The second section describes the advanced machine learning classification algorithms that were used to model classifiers.

2.6.1 Outlier detection

Barnett and Lewis [35] defined outlier as “an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data.”

For centuries, outlier detection has been used to detect anomalies and to remove them in some cases. Outliers exist in several forms, such as natural deviations in populations, mechanical faults, changes in system behavior, fraudulent behavior, or human error [76].

Two popular anomaly detection techniques were used in this research to detect outliers, and they are introduced in the following.

- Interquartile Range (IQR):

  For outlier detection, using the standard deviation around the mean is problematic because both the mean and the standard deviation are sensitive to outliers [111]. Instead, the IQR is used since it is not affected by outliers and rely on the median.

  The interquartile range (IQR) is ”a number that indicates the spread of the middle half or the middle 50% of the data. It is the difference between the third quartile (Q3) and the first quartile (Q1).” [81]
\[ IQR = Q3 - Q1 \]

To determine the quartiles, the median (Q2) has to be found first. The first quartile (Q1) is the median of the lower half of the data, where the third quartile (Q3) is the median of the upper half of the data.[81]

The box plot (a.k.a. box and whisker diagram) is “a standardized way of displaying the distribution of data based on the five number summary: minimum, first quartile, median, third quartile, and maximum.”[1]

Figure 2.3 shows an example of a box plot of a normal distribution \( N(0,2) \) that describes how the quartiles are divided within the dataset.

![Box plot of a normal distribution N(0,2)](image)

Figure 2.3: Box plot of a normal distribution \( N(0,2) \) [11]
The IQR is one of the statistical methods used to detect outliers [76], [162], and [165]. Laurikkala in [105] used a heuristic of 1.5 * inter-quartile range beyond the upper and lower extremes for outliers. Box plot “makes no assumptions about the data distribution model but are reliant on a human to note the extreme points plotted on the box plot.” [76]

- **K-nearest Neighbor**

The K-nearest neighbor algorithm is a well-known algorithm that uses distance metrics for outlier detection as in [138]. The Euclidean distance is a common choice for the distance based algorithms which were used in this research. The greater the distance of an instance to its neighbors, the more likely it is to be an outlier. [25]

Ramaswamy et al. [138] proposed an optimized K-nearest neighbor algorithm that produced a ranked list of potential outliers. A point p is considered to be an outlier if there are no more than n - 1 other points in the dataset that have a higher distance to mth neighbour where m is a parameter specified by the user [76]. Their proposed algorithm was used in this research to develop a k-nearest neighbor outlier detector. The detailed implementation and results are presented in Chapter 6. Numerous research studies have used the K-nearest neighbor algorithm to detect anomalies such as in [94], [155], [93], and [128].

28
2.6.2 Machine learning algorithms

Lee, Stolfo, and Mok [108], from Columbia University, NY, stated that “the intrusion detection can be thought of as a classification problem: we wish to classify each audit record into one of a discrete set of possible categories, normal or a particular kind of intrusion.”

In this section, four different machine learning algorithms are represented that were used in this research for intrusion detection. They are briefly described here, and more details about their implementation are given in Chapter 6.

- **Artificial Neural Networks (ANN)**

  There are various numbers of artificial neural networks algorithms. *Multilayer perceptrons* is one of the most popular types of neural network that is used in many applications such as intrusion detection. It was chosen to be used in this research for many reasons: research has shown that it is an effective alternative to more traditional statistical techniques [142]; it also can be trained to approximate virtually any smooth, measurable function [79]; it is not concerned with data distribution and makes no prior assumptions about that; it models highly nonlinear functions; and it can be trained to accurately generalize when presented with unseen data in the testing data [66].

  The multilayer perceptron is about a model of neurons (often called a node) that are interconnected (as shown in Figure 2.4) and that represent nonlinear mapping between an input vector \((i)\) and an output vector \((o)\). A node receives
an input from other nodes and computes an output. Each input is associated with weight \( w \) to produce an output by applying a function \( f \) to that weighted sum.

![Diagram of a multilayer perceptron with two hidden layers](image)

**Figure 2.4**: A multilayer perceptron with two hidden layers [66]

The multilayer perceptron learns the model from the training data using an algorithm called backpropagation. The Error data at the output layer is back propagated to earlier ones, which allows incoming weights to these layers to be updated [119]. A summary of the backpropagation algorithm training process is listed as follows [66]:

1. Initializing network weights
2. Presenting the first input vector from the training data to the network
3. Obtaining an output by propagating the presented input vector through the network

\[ i = [i_1, i_2, i_3] = \text{input vector} \]
\[ o = [o_1, o_2] = \text{output vector} \]
4. Calculating an error signal

5. Propagating the error signal back through the network

6. Adjusting weights to minimize overall error

7. Repeating the previous steps for the next input vector until a small error rate is reached.

As discussed above, the multilayer perceptron algorithm has many advantages that makes it attractive for use in many applications. Researchers have used it to develop intrusion detection systems as in [68], [46], and [67]. Although it has many advantages, it is slow in terms of building the classifier model, due to its complex process and its involvement with many nodes and layers. More details in Chapter 6.

2.6.3 Decision Tree C4.5

A decision tree is “a hierarchical model for supervised learning whereby the local region is identified in a sequence of recursive splits in a smaller number of steps.” [27] Its data structure follows the divide-and-conquer strategy. It is a well-known algorithm that can be used for classification and regression.

A decision tree is composed of internal decision nodes and terminal leaves as seen in Figure 2.5. Each decision node, $m$, implements a function, $f_m(x)$, with discrete outcomes to label the branches based on the function output.

Figure 2.5 illustrates an example of a decision tree and its corresponding
dataset. The oval shapes represent the *decision nodes*, and the rectangular shapes represent the *leaf nodes*. The univariate node is split along which axis occurs orthogonal to each other [27]. There is no further split after \{x_1 < w_{10}\}, and the subset is considered pure.

There are many decision tree algorithms, and the most well-known algorithm to build trees is the C4.5 algorithm which is an extension of the ID3 algorithm proposed by Quinlan [137] [98]. The C4.5 decision tree algorithm was chosen in this research to develop a classification based intrusion detection system. Many researchers have developed intrusion detection systems using decision tree algorithms, as in [33], [153], and [148]. Also, some researchers have combined using the decision tree algorithm along with other algorithms such as Support Vector Machine (SVM) as in [126], or Naive Bayes as in [64], which produced more effective intrusion detection systems as shown in their results. More details on how the algorithm builds the classifier and its performance is shown in Chapter 6.
2.6.4 Random Forests

The *Random forests* algorithm was developed by Leo Breiman [42], who described it as “a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest.”

The random forests algorithm adds an additional layer of randomness to *bagging*, which was proposed by Leo Breiman in [41]. In bagging (Bootstrap aggregation)\(^3\), multiple trees are fit in to subsampled data where the prediction is calculated by averaging the majority votes of each tree’s response [73], as shown in Figure 2.6.

![Random Forest Diagram](image)

Figure 2.6: A random forest that is composed of a set of B decision trees \((tree_1, tree_2, ..., tree_B)\), with class labels for each tree \((k_1, k_2, ..., k_B)\) to be used to get a \(k\) voted class label [118]

---

\(^3\)the *bootstrap* technique is a statistical method that estimate a quantity from a data sample [43]
Each node in the standard trees is split using the best split among all variables. However, in random forests, each node is split differently by using the best split among a subset of predictors that is picked randomly at that node; which helps to overcome the overfitting problem [113]. Thus, this is one of the main advantages of the random forest algorithm, in addition to its ability to handle high dimensional data. [69]

Random forests was first used in intrusion detection by Zhang and Zulkernine in [169], when they performed a network-based misuse and anomaly detection. Many other researchers used it for the same purpose, as in [170], [156], [169], and [62]. In this research, the random forest algorithm also was used to develop an intrusion detection system. The details of how the classifier was trained and how it performed are presented in Chapter 6.

2.6.5 Support Vector Machines (SVM)

The support vector machine algorithm [158] history goes back to the seventies. In the nineties, the SVM was developed, based on results from statistical learning theory [143]. The principle of the SVM algorithm is to derive a hyperplane, which maximizes the separating margin between the positive and negative classes [63]. Figure 2.7 shows the support vectors of the two classes, (+) and (-), are the data points that are closest to the hyperplane.

The SVM algorithm becomes popular for its generalization ability, especially for its high number of features, \( m \), with low numbers of data points, \( n \) [44]. However, training the SVM with a dimensional quadratic programming (QP) prob-
lem involves large matrix operations that result in large numbers of computations which lead to slow performance [98]. John Platt [136] introduced an algorithm called *Sequential Minimal Optimization (SMO)* that solved this issue. It breaks a large QP problem into a series of very small QP subproblems that are solvable analytically. The computation time of SMO is dominated by kernel evaluation; hence, the use of kernel optimizations can be accelerated. [74] Also, many enhancements were applied to the SMO algorithm which increased its performance even more than Keerthi et al. did in [85].

The SVM algorithm in general has been used for decades for both anomaly ([80], [89], [147], [115]) and misuse detection ([92], [78], [28]). The SMO algorithm was used in this research to develop an intrusion detection system due to its accuracy and performance. .


2.7 Attacks on cloud computing

The computing environments suffer from numerous type of attacks. The cloud environment in particular suffers even more due to its nature of sharing the resources among users. A Virtual machine in the cloud environment shares hardware resources with other VMs. A container in the cloud environment even shares the same operating system with other containers. Although there are isolation technologies involved, some shared resources remain, whether it is software or hardware which expose the entire environment to potential risks. In this section, some attacks that threaten the cloud environment which were used in this research to generate abnormal data to be used in the testing phases, are briefly explained as follows:

- **Network eavesdropping**: attackers snoop on connections between VMs or containers.

- **Brute force attacks**: an attacker attempts to login to the system by using a dictionary or large list of words.

- **SQL Injection**: an attacker inserts malicious SQL statements into entry field to be executed.

- **Port scanning**: an attacker scans the ports in the system and tries to expose a service through an open port.

- **Distributed Denial of Service**: an attacker tries to exhaust the system by flooding requests from many sources, attempting to make the system unavailable for legitimate users.
• *User to root attack*: an attacker tries to get into the system using legitimate user credentials that she/he got by sniffing for passwords. Then, she/he can gain root level access by exploiting vulnerabilities.

These attacks are meant to be used in this research to try to maximize all potential risks against the cloud environment, since they affect directly the security goals of availability, integrity, and confidentiality that were discussed earlier. The next chapter describes the literature review, presented the most related systems, and how the related work differ from this research’s proposed system.
Chapter 3

Literature Review

Throughout the past decade, researchers have developed a large number of hypervisor based intrusion systems. Some of these proposed systems were developed to work in the cloud environment or they could be implemented in the cloud environment. The presented systems are combinations of HIDS/NIDS and signature/anomaly based or hybrid systems. For the detection techniques, statistical approaches, rule based, or machine learning techniques were used.

Kholidy et al. [91] proposed a hierarchical and autonomous cloud based intrusion detection system where the framework continuously monitors and analyzes the event on the system. It was designed to secure VMs and back-end servers. The framework contains a controller that receives security parameters with their risk impact factor and selects the appropriate response to protect the system against attacks. The authors argued that the system is has capabilities to provide self-resilient and fault tolerant.
Kourai and Chiba [99] developed HyperSpector, which is a virtual distributed monitoring environment that achieves secure intrusion detection in distributed computer systems. The IDSs that protect distributed systems could increase the insecure points in the protected system. Thus, the authors designed HyperSpector to overcome this issue without adding hardware. They used virtualization technology to isolate the servers from their IDSs. The IDS VMs connects to a server VM using a virtual network. The proposed system provides three inter-VM monitoring mechanisms: software port mirroring, inter-VM disk mounting, and inter-VM process mapping. The system is resistant to active attacks and mitigates the passive ones.

Li et al [112] proposed a distributed neural network based IDS for the cloud environment. It is designed to target VMs that were tested against a large dataset of VM network traffic. They achieved high accuracy, over 99%, in detecting attacks in the cloud infrastructure.

Pandeeswari and Kumar [129] proposed Hypervisor Detector which is an anomaly based detection system that works at the hypervisor layer. It uses both algorithms Fuzzy C-Means clustering and Artificial Neural Network to increase detection accuracy. The C-Means algorithm creates cluster subsets, and the ANN algorithm trains each cluster by aggregating modules. They used the KDD cup dataset 1999 for their experiments, and claimed that the proposed system reached a detection rate of over 97% but with overhead performance, due to its complexity.

For a finite state machine based approach, Kumar et al. [101] proposed a cloud-based intrusion detection system that uses the Hidden Markov technique to model the user behavior transitions over certain periods of time and detects intru-
sions based on the probability of the behavior. It generates the future probability based on the previous ones. The model seeks behavior continuously and reports it to the filtered network that is attached to the firewall. The possible cases of behaviors are as follows: high profile which means the observed pattern has a high probability of matching the baseline model of that specific user; the middle profile means that the observed pattern has partial matching probability; and low profile means the observed pattern has a low probability of matching the baseline model. All these matching processes are based on a predefined threshold. The authors claimed that the system could catch an intruder even if it had a legitimate ID and password for the network.

Srinivasan et al. [152] designed a system called eCloudIDS that detects anomalies in VMs running in the public cloud environment. The proposed system was designed with hybrid two-tier engines, uX-Engine, and sX-Engine. The uX-Engine uses an unsupervised machine learning algorithm (Self Organization Map) for classification. Their results showed a detection rate of 89% with a minimal false alarm rate of 2%.

Some of the developed systems were implemented by applying an old, efficient method to the cloud. For example, Gupta and Kumar [72] used Forest et al’s [65] method where they proposed an Immediate Syscall signature structure based technique that detects malicious executions in the cloud environment. It creates a database of system calls that are formatted using a key-value structure where the key is the unique system call name, and the value is the immediate sequence of system call that follows it during program execution. The detection system finds any mismatch from the baseline and an alarm identifies it as an
anomaly sequence. It is efficient in terms of complexity and the authors argued they achieved 98% detection rate. The same authors in [71] used the Malicious System Call execution Detection (MSCD) approach which creates program-wide MSCD databases at each VM and one at the cloud manager. In this approach, the MSCD at the client VM is compared to the baseline MSCD database, and it looks for mismatches to detect anomalies at the system level.

Another example of using an old method was using the frequency-based approach (T-STIDE [164]) with a machine learning technique to detect anomalies. Kang et. al [84] used it in their research with their proposed feature vector bag of system calls where the order of the system calls is ignored. This approach was tested with several machine learning algorithms and provided good results. It was also used without machine learning detection being involved, and provide good results too, as used by Alarifi and Wolthusen in [26]. They used the “bag of system calls” approach to detect anomalies in the cloud environment by calculating the difference in frequencies of sequences between the tested samples and the target VM normal behavior profile. Also, the method was used for containers (running on localhost) intrusion detection to detect anomalies, as in [24] and it provided good results as well. The “bag of system calls” method also was used as a part of the enriched data representation mechanism that was proposed in this research as explained in Chapter 4.

The intrusion severity analysis approach was proposed for the cloud environment by Arshad et al. [32] at domain 0, as shown in Figure 3.1. For the proposed severity approach, they used the security requirements for a virtual machine proposed in [31] that categorized by three security attributes, as shown in
Table 3.1. The Misuse detection approach was used to detect suspicious system calls of known attacks in the monitored VM. Then, those system calls would be transferred for further analysis by the severity module that used decision tree algorithm for intrusion detection. Over 90% detection rate was achieved by their system.

![Figure 3.1: An Intrusion Severity System](image)

Virtual Machine Introspection (VMI) was introduced to provide security and reliability for virtual machines [166]. It is used in the cloud environment when the provider needs to monitor the VMs from hypervisor [120]. The intrusion
Table 3.1: Security Requirements for a Virtual Machine

<table>
<thead>
<tr>
<th>Security Attributes</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity</td>
<td>Workload State Integrity</td>
</tr>
<tr>
<td></td>
<td>Guest OS Integrity</td>
</tr>
<tr>
<td>Availability</td>
<td>Zombie Protection</td>
</tr>
<tr>
<td></td>
<td>Denial of Service Attacks</td>
</tr>
<tr>
<td></td>
<td>Malicious Resource Exhaustion</td>
</tr>
<tr>
<td></td>
<td>Platform Attacks</td>
</tr>
<tr>
<td>Confidentiality</td>
<td>Backdoor Protection</td>
</tr>
</tbody>
</table>

detection systems of the VMI (VMI-IDS) are deployed at the Virtual Machine Monitor (VMM). One of the VMI intrusion detection approaches was implemented by Payne et al [132] where they developed a VMI framework called *Lares* that does active monitoring that places a hook inside the monitored system. The execution would be interrupted when that hook was reached and control was passed to the security tool. Also, the monitoring process can take place outside of the monitored system. The security application is split into two VMs, untrusted guest VM and security VM, which is part of the the trusted computing base (TCB). The integrity of the hook is protected by a special memory protection mechanism. The guest VM contains the hooks and a special crafted trampoline code\(^1\) to achieve the active control and monitoring capabilities given by Lares. The security VM contains the core of the active monitoring when decision making takes place. After making the decision, the security application sends the decision back to the guest VM to be enforced.

*Collabra* is also a distributed VMI-based IDS that was proposed by Bharadwaja et al.[40] for maintaining the security of cloud based virtualized environments.

\(^1\)This code is to pass events signaled by the hooks to the hypervisor
It acts as a filtering layer that is integrated with each VMM and scans through hypercalls\(^2\) for integrity checking. It is distributed between multiple machines and acts concurrently over VMM networks to detect attacks and prevent unauthorized access to the VMM and the host.

Hypervisor introspection (HVI) was introduced to secure the hypervisor from being controlled by an attacker. Carbone et al. [47] developed an intrusion detection hypervisor introspection-based that used nested virtualization. They introduced GuardHype, which is a layer of security underlying the hypervisor that enhances its security and prevents VM-based rootkits (VMBRs). The main two tasks of GuardHype are controlling third-party hosted hypervisors to prevent users from misusing virtualization, as well as supporting the OS’s use of virtualization for security purposes. Another research study on hypervisor based IDS that takes advantage of virtualization technology was done by Zhang et al. [168]. They proposed a transparent approach that protects the privacy and integrity of VM on a virtualized infrastructure. The proposed system separates the resource management VMM from security protection in the virtualization layer. Nested virtualization [70] [37] was used by the security monitor to provide protection to the VMs. Thus, it handles the complexity of managing the VMs in the cloud without breaching the security of users’ data inside the VMs.

Hybrid intrusion detection approaches have been developed by many researchers who they have a system that is misuse/anomaly based at the same time.

---

\(^2\)Xenproject wiki defines hypercall which is a software trap from a domain to the hypervisor, just as a syscall is a software trap from an application to the kernel. Domains will use hypercalls to request privileged operations like updating pagetables. [20]
For example, Modi et al. [124] developed a framework that integrates an NIDS in the cloud environment to reduce the impact of network attacks. The framework was implemented using Snort and Decision Tree (DT) classification. The hybrid system uses Snort to detect known attacks, whereas the DT algorithm is used for anomaly detection. The system passes the captured packets to Snort, and if any matches found, an alert would be raised. If not, it passes them to preprocessing, then to the DT classifier that detects anomalies, based on the stored behavior base. They achieved over 84% accuracy for the NSL-KDD dataset and 96% for KDD dataset. Also, the authors in [125] used the same module, but one that was composed of Snort and a signature apriori algorithm that generates new rules that were added to the Snort configuration file to improve efficiency.

An intrusion detection system for Infrastructure as a Service (IaaS) cloud was introduced by Tupakula et al. in [157]. It secures users’ virtual machines from several types of attacks generated by VMs and could differentiate which attack traffic came from which VM, even if the VMs shared the same IP. Their Virtual machine Intrusion deteCTOR’s (VICTOR) main purpose is to determine the malicious entity that generates the attack flow and then to dynamically isolate that entity. The system is composed of several important components that work together. The OS library and repository (OSLR) component contains generic knowledge about the operating system of the VMs when they were first initiated at the IaaS provider. Since the OSLR has the resource details of the VMs, the Detection/prevention Engine uses those details to determine attack signatures for the VMs, since it has a database that contains the known attack signatures. VICTOR is a hybrid system, so, it contains an anomaly based engine which applies machine learning techniques to the OSLR data to detect malicious behavior from
A trusted monitoring framework for cloud platform was designed and implemented by Zou et al. in [172]. The framework provides a chain of trust where the untrusted privileged domain is excluded. That can be done by deploying an independent guest domain for monitoring, as well as utilizing the trusted computing technology to ensure the integrity of that monitoring system. They proposed the monitoring driver which contains the instructions that performs the OS information on the monitored VM. It is dynamically loaded when launching the corresponding VM on the cloud platform. The monitoring VM contains different shapes specified for each type of monitoring driver that correspond to different guest OS. Also, the framework consists of a management VM (MVM) entity that controls and manages other VMs. Moreover, an Event Sensor entity exists in the VMM to intercept certain events where that interception can be configured by each driver. The authors argued that adopting the trusted computing technology to the system could solve the trust issues between cloud tenants and providers. They implemented their proposed framework on a cloud platform called OpenNebula with moderate performance overhead.

Shelke et al. [146] proposed a multi-threaded distributed cloud IDS system that handles large data packets, and analyzes, and reports them efficiently. It detects intrusions by integrating knowledge and behavior analysis. Their model consists of three modules: the capture & queuing module, the analysis/processing module, and the reporting module. The in-bound and out-bound data packets (ICMP, TCP, IP, UDP) are received by the capture module which sends them to the analysis/process module to be analyzed against a signature base and pre-
defined set of rules. The processes in the capture module can be multi-thread that work concurrently. Finally, the alerts would be sent to the reporting module to generate reports. The authors argued that the proposed system handles high volumes of data. However, the research does not show implementation of the proposed system or even proof of concept.

Having described several of the proposed cloud based intrusion detection systems, many others are not presented because there are so many of them. Some of the examples that were not listed and which are related to what we presented are [160],[106], [104], [139], [117], [58], [87], and [114]. Additional lists are included in various surveys that list a number of different types of cloud IDSs, such as [123], [120], [131], and [50]. The following section describes research and tools that is similar to our proposed intrusion detection system.

3.1 Related Work

Some of the previous research were regarding to VM and hypervisors and have limitations, such as requiring knowledge about the VM, having issues with the data representation, or having issues in terms of complexity. The hypervisor based and VMs intrusion detection are related technologies to the containers since they work on hypervisors and work in cloud environments, as well.

Abed et al. [24] developed an anomaly intrusion system for containers using bag system calls for learning containers' behavior in order to detect anomalies. It was not implemented in a cloud environment, which is critical, as was
discussed in section 2.3.1, showing that the cloud environment is different from the traditional environment and needs to be considered. This research advanced the intrusion detection systems for containers significantly, in terms of the data representation used, the detection methods, and the implementation process, in addition to implementing it in a real cloud environment, as described in detail in Chapters 4, 5, and 6.

Mattetti et al. [116] developed a framework for securing Linux containers and their workloads by constructing rules for the normal activities of the monitored container. The developed tool constructs a profile of the kernel security module for the monitored container by tracing the kernel operations. Then, it translates those operations to AppArmor \(^3\) rules. This tool is not a complete HIDS, but it can be integrated with the intrusion systems. However, using the AppArmor module as the rule extractor could affect its performance as well as increase the false positive rate by blocking numbers of legitimate operations. Also, this tool is intended to be applied to a cloud environment, but it was not tested in the cloud which could present unexpected results since the cloud involves interactions with numerous components besides the host interaction with the containers. The authors justified their choice of tracing the kernel functions instead of system calls because it is impractical to use system calls with the high volume of the generated system calls from the containers. However, the opposite is proved in this dissertation, where the system calls can be used as a data source to develop an IDS for the container and it works well in real-time in a cloud environment.

---

\(^3\)AppArmor is a Linux Security Module that provides mandatory access control (MAC)
Barlev et al. [34] developed a misuse HIDS of a server running Linux containers by intercepting kernel functions that have the potential of being invoked by an attack. Each interception is considered an “event” that is passed to the event processor to be analyzed. The stream of events are the input of the system attack detection logic. The system monitors the server behavior and creates a customized security policy which consists of rules and actions to be followed. This developed system is different from the one described in this dissertation in many ways. Barlev’s system was not implemented in a cloud environment. Also, the explanation of how the policy was created was vague. Also, Barlev et al. argued that they chose carefully a subset of system calls to trace, but they did not mention the criteria for choosing that subset. In addition, learning the behavior of the server lasted for several days, which is a very long time and cannot be suitable for a cloud environment since it needs to work in real-time. Also, during the learning process, all activities are passed to the system, whether it is normal or not. With a long period of learning process, having abnormal activity during that period could be learned as normal, which creates a dilemma.

For commercial products, there is an auditing tool called Falco [10] developed by Sysdig that runs in user space and can be used to monitor Linux hosts, including containers based on the Sysdig monitoring technology. Falco can be integrated with external systems, such as Docker, Kubernetes, Mesos, etc. The system of this research was developed for academic purposes and shows all the models and technologies used for that purpose, step-by-step. Although Falco is used for commercial purposes and targets different goals than this dissertation’s proposed system, it needs further investigation to compare Falco results with this system in terms of detection ability as well as performance. Falco consists of sev-
eral features besides analyzing system calls which might affect its performance. This is an area for future work.

The next chapter describes this dissertation’s proposed enriched data representation in detail, and how it is constructed, in addition to how it has been used by the detection algorithms.
Chapter 4

Enriched Data Representation

Chapter 2 described in detail the difference between the two types of intrusion detection which are anomaly based or misuse based. The anomaly detection (e.g. outlier detection) is a baseline (normal profile) of the host behaviour and any deviation from that baseline is considered an intrusion, whereas the misuse uses patterns (signatures) of known attacks to identify intrusions. In this chapter, the enriched data representation is described in detail.

The proposed data representation was designed to work for both anomaly and misuse intrusion systems as shown in Chapter 6. It was designed to be constructed as quickly as possible because the cloud intrusion detection system has to work in real time as discussed in Chapter 2. Thus, the decision to determine intrusions has to be fast as well.

The structure of this chapter is as follows. The data source used in
the research is described first, followed by an explanation of how that data was collected and pre-processed. The enriched representation mechanism is explained next in detail, as well as explaining the motive for its design by mentioning the issues with the previous representations. At the end of this chapter, the process of how the feature vector was constructed is explained and how that representation was used to create the normal profile which was used to extract the features to construct both normal and abnormal samples.

4.1 Data Source

As described in Chapter 2, the system calls were first used by Forrest [65] as a data source to build an intrusion detection system. After that, many researchers, as in [77], [164], [55], and [100], used the system calls as the data source to build intrusion detection systems for Linux applications, virtual machines, and Linux containers. Throughout that research, the system calls proved to be a preferred source of data since it is able to access the core kernel functions [54]. Forrest and her team used the system calls as the data source, assuming that the programs have consistent behaviors where they can be used to build a profile. Also, they assumed that the attacks would generate different behaviors from the normal behavior. Using the same assumptions, this dissertation’s author assumed that Linux containers running in a cloud environment have consistent behaviors since the container runs one program (MySQL, Apache, etc.)
4.2 Data Collection

There are many public datasets that were generated for testing intrusion detection systems. However, a search of recent publications found no research that designed and implemented a host-based intrusion detection for containers in a real cloud environment. Thus, building a cloud environment is needed to collect real data as the containers running in the cloud environment might have different behaviors, than the ones running in the local host because the container running in a cloud interacts with many resources that also interact together, which might change the containers’ normal behaviour. It was shown in Chapter 2 how the cloud performed differently than the traditional computing environment.

Using the existing popular datasets that might work for containers since they are still Linux systems was not considered in this research for several reasons: First, there are no new datasets, since the new Linux system kernels contain new system calls that were not in the old Linux kernels, and some system calls were removed from the new kernels. Such changes in kernels could affect the results. Second, the attacks used in the old system were old and there are more sophisticated attacks today. Third, gathering the data from containers working in a real environment is the best choice, as they might act differently than working in local hosts since there are many cloud components involved and they interact with the containers. The only way to test this hypothesis is to build a real environment that uses the current fresh updated versions of Linux containers kernels in cloud environment and use real and smart attacks that are currently used. Thus, Linux containers running on a cloud system were built, implemented, and tested at the Security Lab at the Florida Institute of Technology.
There were assumptions before collecting the data: the container from the official docker hub website [8] contains no malicious code nor does it have any vulnerabilities. Thus, after building the cloud environment (see The Environment Design section 5.1 for more detail), all the system calls of all processes generated by the targeted container were gathered using tracing tools Strace [15] and Sysdig [17] for the different experiments described in the following sections. The trace includes all the Process Identifier processes (PID) and their child processes in order to build a normal profile that represents the container for its normal operations. Figure 4.1 shows a sample of the Strace gathered for one container.

![Strace sample](image)

Figure 4.1: Strace sample

### 4.3 Data Pre-processing

The tracing tools provide many details about the invoked system calls, including a list of paths, a list of file descriptors, time execution, error code, arguments return...
value, and much more. In this research, only the system call names were used, without considering their arguments and return values or any other data in order to construct the sequences as described in detail in section 4.6. The reason for considering a few details about the system calls (only the names in this case) was to test if it was possible to build a model that represents a container in a cloud environment quickly and efficiently as these requirements are needed to build an intrusion system that works in real time.

4.4 Sliding Window Technique

Using the sliding window technique and system calls frequencies within each window to produce sequences of system calls in order to generate the normal profile of the targeted applications was introduced in [164]. In this research, the sliding window technique was used, along with other factors to enrich the representation of the data in order to design efficient, fast, simple, and intelligent intrusion detection systems that give better results for containers in cloud environments. A window size of 6 was used in STIDE experiments [164] and other research [65] and [77] and reported to be the best window size option that gave better results. However, a window size of 10 was chosen in this research because it gave more accurate results with low overhead and a fast learning process as shown in [26] and [24]. Figure 4.2 shows the how the window size of 10 was built. The part above the red line showed how the window was filled and the bottom part is the window advancing.
The intrusion detection system, that uses system calls as inputs, is formally defined as follows:

Let \( \Sigma = \{s_1, s_2, s_3, ..., s_m\} \) be a set of system calls where \( m = |\Sigma| \) is the number of system calls. Dataset \( D \) is defined as a set of labeled sequences:

\[
\{< Z_i, c_i > | Z_i \in \Sigma^*, c_i \in \{0, 1\}\}
\]

where \( Z_i \) is an input sequence and \( c_i \) is the corresponding class label where 0 denotes the “normal” label and 1 denotes the “abnormal” label. The aim of the learning algorithm for the dataset \( D \) is to find a classifier \( h : \Sigma^* \rightarrow \{0, 1\} \) that maximizes the defined evaluation criteria [84].

The bag of system calls was introduced in [84]. It used a popular model
in text classification called Bag of Words [121]. The bag of system calls method converts the input sequences to the bag of system calls. Thus, the ordering of the system calls is lost and it only preserves the frequency of each system call within that sequence. The representation is formally defined as an integer-frequency based method where a feature is defined as an ordered list, \( X_i = \langle c_1, c_2, c_3, ..., c_m \rangle \) where \( m = |\sum| \) and \( c_j \) is the frequency of system call \( s_j \) in the input sequence \( Z_i \) [84].

The bag of system calls method was applied in addition to the sliding window technique to design more effective intrusion detection systems and it performs better than the traditional techniques that used contiguous subsequence methods in Forrest’s research [65] and [77].

4.5.1 Issues with the Previous Representation

Most of the IDSs that model the process behavior used the contiguous sub-sequences as inputs, such as in [164], [65] and [77]. Whereas the bag of system calls representation was used as inputs by several researchers, such as [84], [26] and [24]. For detecting the anomalies, the authors in [24], for example, used simple detection algorithm which compared the tested sequences to the normal profile. If that sequence was not in that profile, a mismatch would be declared, and if mismatches exceeded a defined threshold within one epoch\(^1\), a signal of anomaly would be raised. This simple method that was used in several research studies suffers from several drawbacks, as follows:

\(^1\)The large traced files were divided into small files (epochs) where each epoch contained 5000 system calls. The same size of epochs was used in this research as well.
• The sequences that were in the normal profile with “any” frequencies (less or higher than the threshold) but not found in the tested sequences would not be tested.

• The abnormal sequences with frequencies less than the determined threshold would not be considered as anomalies.

• The sequences used for testing with frequencies higher than the determined threshold and found in the normal profile would not be considered as anomalies even if they were. That was because the detection method does not check for the existing sequences, only the mismatched ones.

• Not all the normal sequences can appear within a certain period or epoch. Two normal sequences or more (with frequencies that do not exceed the defined threshold) could be anomalies, especially if they never happen together at the same epoch in the normal profile.

• Some normal sequences should appear at certain ranges of frequencies where having less of them in certain times can be an anomaly activity.

All the above issues would not be solved by the mentioned detection algorithms. Using only the sequences that exist already in the normal profile is not enough to identify anomalies if the tested sequence has unusual frequencies that are never seen during training of the normal model. Also, it could be a case of normal sequences with familiar frequencies but which happen to be near each other when they should not be or vice versa. In that case, they would not be considered as an anomaly, even they were. Also, the detection system that uses
the average as an indication of mismatch would not consider it as an anomaly because the frequencies were familiar and would not exceed the threshold.

Setting up a threshold for detecting anomalies is a difficult task. So, the anomaly sequences that fall below the threshold would not be discovered and they could cause harm. Some smart attacks do not register many sequences or frequencies and act as normal activities. Also, if the detection method does not check for the frequencies of the new sequences even if they existed in the normal profile, it could miss detecting anomaly sequences even if they were in the normal epoch files but with higher frequencies.

The Linux operating system has over 400 system calls. They should not all be considered for the system call sequences, as that would affect the performance and would use more storage. The research of [26] and [24] as examples, used only the system calls that appeared in the experiments. They read the traced file epochs and used only the used system calls. They registered the rarely used system calls in the “other” category. The system reads the epochs one-by-one until reaching profile stability, where the normal profile represents the targeted VM or the container. The profile contains the unique bag of system calls sequences and their frequencies. During the training, the sequences used as inputs and the detection algorithms find mismatches between the normal profile and the tested sequences and raised anomaly signals if it exceeded the defined threshold, as described earlier.
### Table 4.1: Threat Level Classification

<table>
<thead>
<tr>
<th>Threat level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Allows full control of the system</td>
</tr>
<tr>
<td>2</td>
<td>Used for a denial of service attack</td>
</tr>
<tr>
<td>3</td>
<td>Used for subverting the invoking process</td>
</tr>
<tr>
<td>4</td>
<td>It is harmless</td>
</tr>
</tbody>
</table>

4.6 Enriched Data Representation

In this research, more advanced methods have been used to enrich the representation that can address the above issues. It is a combination of existing methods and new ones which have been introduced in this research. Not all the system calls that appeared while creating the normal profile was used as in [24]; the security requirements using REMUS proposed in [39] were applied where only the high threat system calls were included, as in [38], [26], and [32]. Also in this research, the system calls that appear very frequently\(^2\) were used even if they had low threat level because the containers’ normal operations mostly rely on them. Missing those frequent system calls could indicate that there are malicious activities that are taking over.

Bernaschi et al. [39] classified the system calls into four difference threat models, as shown in Table 4.1. In this research, only level 1 - 3 high threats were included in addition to the frequent used system calls even if they were not classified as a high threat for the reasons mentioned earlier.

\(^2\)The pre-defined threshold is set by experiments as they vary from one container to another, depending on the type of application that runs in that container.
Since Forrest and her team started working on developing the intrusion detection system using the system calls as a data source, all the research presented in Chapter 2 showed that the system calls sequences were used as input for the detection algorithms. However, using the sequences as inputs have some drawbacks explained earlier in this chapter. Section 4.6 describes an improved method to enrich the system call sequences representation that helped to detect more advanced attacks and overcome issues introduced by the previous representations. The new representation maintains fast learning with low overhead performance.

4.6.1 Constructing the Feature Vector

The system designed in this research used ideas similar to the ones in the previous research where the system goes through system calls epoch files one-by-one to build the normal profile for the target system. Then, the bag of system calls would be constructed when applying the criteria, described earlier, to which system calls would be used to construct the bag of system calls sequences. Each bag of system call sequence was used as a feature in the feature vector. Building the normal profile that represents normal operations of a container should be constructed as fast and accurately as possible because it needs to work in real time in the cloud environment.

As assumed in [65], the application’s operations have certain patterns and form a normal behavior of that application. Considering these activities when they work together and treating them as chunks of sequences is more accurate than just looking only to one sequence of these activities to judge if the activity is normal or
not. When looking at the system calls that happen during a certain specific time gives more reliable decisions. Representing the sequences as features is shown in Chapter 6. In addition to the sequences features, an additional feature was added to the features vector that counts the new sequences that were not seen during building of the normal profile. Even though the features by themselves provide promising results with high detection and low False Positive rates, this additional feature improved the accuracy and the detection rate as described in the discussion section 6.7.

### 4.6.2 Building the Normal Profile

Since the application container does a specific task, it is assumed in this research that it is possible to learn its normal activities from the generated system calls, as proven by Forrest et al. [65] and all other researchers that followed their methods. Learning normal activities of the whole host (e.g. a server, host, or virtual machine) is very difficult due to the complexity of their operations. Also, their operations are difficult to be anticipate because it varies from one user’s usage to other’s. Also, running multiple applications would make it even more difficult, unless it is dedicated for one single application with limited operations. Thus, building the normal profile for a container running in the cloud was to be built relatively fast and to represent the container behavior well.

The normal profile was built in this research for two purposes that allows the introduced intrusion detection to work for both anomaly and misuse intrusion detection systems. For the anomaly detection system, the profile was built to
create a normal database that represents the container that worked in the cloud environment and was used as an alarm for any deviation (or outliers) happening during testing, marking them as anomalies. For the misuse intrusion detection systems, the profile was built to learn the normal sequences from the container normal activities and to use them as features (in addition to the distinct feature) for the training and testing samples. More details are described in section 4.6.3.

Unlike other research in the past that used the system calls as features to feed the detection classifiers, in this research the sequences of the normal profile were used as features for the detection algorithms. Some of the issues of using the system calls as features were explained earlier. The new representation of the features has shown promising results which improve the detection against sophisticated attacks and unusual normal activities.

### 4.6.3 Forming the Samples and the Attacks Patterns

After running the detection system through the system calls to learn which sequences could represent the normal behaviour of the container, it starts forming the samples of the normal activities where the frequencies of each sequence (feature) of the epoch files (samples) were registered, based on the normal profile features. Each sample would register the total of bag of system calls (n-grams) as follows: If $X$ is the total number of system calls in one epoch, $S$, the number of n-grams ($N$) for epoch $S$ is computed as follows [21]:

$$N_{gram,S} = X - (N - 1)$$
The same process were performed for the attacks patterns where the attacks activities were registered in feature vectors that mapped to the normal profile features. This proposed system could detect the attacks very accurately due to using this method, as shown in Chapter 6. A python program was written to automate the sample formation of the normal and abnormal samples. The sample formation process was programmed to be done very quickly, as shown in the results as well. The type of normal activities and the attacks are explained in Chapter 5.

As mentioned in Chapter 2, the misuse detection systems only detect the known attacks, not the unknown (or zero-day attacks) ones that were not in the database. However, forming the attacks based on the normal profile features increased the accuracy significantly, as well as detecting the unknown attacks that were not trained by the misuse detection classifiers, as presented in the Chapter 6 as well.

The normal profile feature vector consists of the sequences that happened during normal usage of the container. When forming the attack patterns, only the sequences that were in the normal profile features vector would be considered and counted in new vectors that represent that type of attack. The new sequences that the attack generated would be counted by the distinguished feature that was added to the normal feature vector initially. Forming the normal and abnormal activities based on this method helped increase the accuracy, True Positive Rate, and False Positive Rate when using the machine learning algorithms.

The next chapter describes how the experiment’s environment was designed. It also provides details of the type of activities used to generate the normal
and abnormal samples. Also the proposed framework of the intrusion detection system is described in detail.
Chapter 5

Environment and Framework Designs

5.1 The Environment Design

In order to test the proposed intrusion detection systems realistically, real world data has to be used. Since there is no public data available for containers run in cloud environment, a cloud system was built to run containers and gather their normal and abnormal data. This research’s cloud environment used the OpenStack deployment.

Before describing the design of this cloud environment, brief descriptions of the minimal OpenStack platform components (Figure 5.1) are explained as follows [12]:
The Compute service component (Nova) provides services to support managing virtual machine instances. It facilitates this management by using an abstraction layer that interfaces with the chosen hypervisor for the deployment.

The Object Storage service (Swift) and Block Storage service (Cinder); where Swift provides support for storing and retrieving data in the cloud; Cinder provides persistent block storage for compute instances. It is responsible for the management of the life-cycle of block devices from the creation phase to the release phase.

Networking service (Neutron) provides networking services to users or the cloud, such as managing their IP addresses, DNS, DHCP, etc; the users also have the ability to manage and configure their networks.

Dashboard (Horizon) provides a web-based interface for both cloud administrators and cloud users. Besides the ability to manage the cloud services through the terminal, the dashboard also allows the administrators and the users to provide, manage, and monitor the cloud resources.

Identity service is a shared service that performs authentication and au-
thorization services between all the cloud resources.

*Image service (Glance)* is a service that manages disk-image services, such as image discovery, registration, and delivery services to the Compute service (Nova) as needed.

Having described the services of OpenStack components briefly, Figure 5.2 shows its architecture and how these services components interact with each other.

![Figure 5.2: OpenStack Architecture [12]](image-url)
Besides OpenStack being an open source, free of charge, and used by major companies such as PayPal, one of the main benefits of using it is its flexibility. Also, it allows integration with other services and hypervisors such as Docker\(^1\) to the cloud that was built by OpenStack. Docker containers were used in this research, and the designed intrusion detection was built specifically for those containers. Figure 5.3 shows how Nova interacts with Docker so the Docker image can be uploaded and run in OpenStack.

![Figure 5.3: OpenStack-Docker interactions](image)

The Nova driver embeds an HTTP client that talks, via a Unix socket, with the Docker internal REST API. The driver controls containers and fetches their information using HTTP API. Also, it fetches images from the OpenStack Image service (Glance) and loaded them into the Docker filesystem [13] as illustrated in Figure 5.3. After setting up the above environment, the IP addresses were assigned to the services as shown in Table 5.1.

\(^1\)Docker is a tool designed to make it easier to create, deploy, and run applications by using containers[5]

69
Table 5.1: The main cloud components IP addresses

<table>
<thead>
<tr>
<th>Unit</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ceph RADOS Gateway</td>
<td>10.102.165.150</td>
</tr>
<tr>
<td>Ceph-osd</td>
<td>10.102.165.185</td>
</tr>
<tr>
<td>Glance</td>
<td>10.102.165.158</td>
</tr>
<tr>
<td>Keystone</td>
<td>10.102.165.146</td>
</tr>
<tr>
<td>Mysql</td>
<td>10.102.165.79</td>
</tr>
<tr>
<td>Neutron</td>
<td>10.102.165.99</td>
</tr>
<tr>
<td>Nova Cloud Controller</td>
<td>10.102.165.234</td>
</tr>
<tr>
<td>Nova Compute</td>
<td>10.102.165.205</td>
</tr>
<tr>
<td>OpenStack Dashboard</td>
<td>10.102.165.65</td>
</tr>
</tbody>
</table>

5.2 Data Generation

For the research scope, the application containers are targeted, but an extended study will follow this research that targets the machine containers and will apply the same methodology to test if they are applicable to building an effective intelligent intrusion detection systems like those built for the containers in this research.

5.2.1 Normal data

The official MySQL Docker image was used in this research to test the proposed enriched representation and the intrusion detection framework. Normal activities were launched on the MySQL container to construct the normal behavior profile first. Also, part of whose were used to generate normal samples to train the classifiers models along with the abnormal samples that are described in the abnormal section 5.2.2.
Both manual and automated activities were performed on the container to maximize the chance of imitating the real world activities and data. A real database that is publicly available was used to create the initial schema and tables in the container. Different types of SQL queries, such as select, delete, update, and insert were run on the container with different values in different times. Multiple SQL client users were simulated to connect to the container and run automated SQL queries and different commands concurrently, consecutively, and non-concurrently. The connections were from the host, The Internet, and from other containers in order to simulate real world client users’ activities.

5.2.2 Abnormal data

In order to test the designed intrusion detection systems, abnormal activities should be run against the container to determine if they would be distinguished from the normal activities or not. The same procedure as the normal activities were applied here to generate abnormal activities. Manual and automated activities were used to issue attacks, where some of them used advanced tools to do so. Below is a list the attacks used in the experiments:

- Network eavesdropping to compromise connection
- Brute force attacks
- SQL Injection
- Port scanning
- Distributed Denial of Service
• PING flooding

• User to root attack

Not all the used attacks were easy to detect because they do not significantly impact the system. Some of the attacks make many obvious changes, and they are easy to be detected. Thus, including advanced attacks would give a good indication whether the system is capable of detecting current attacks or not, and whether the system would work well with the complex operations run by the cloud or not.

5.3 The Framework Design

Having discussed the process of how the data were gathered and how the representations were constructed, this section describes and explains how the intrusion detection system framework was designed to form a realistic representation of a container system calls working in a cloud environment. Also, it shows how the misuse and anomaly detection systems detect intrusions. The framework helps the system to detect the intrusions effectively, as shown in Chapter 6.

The designed intrusion system architecture for containers working in real time (or near real-time) in a cloud environment is shown in Figure 5.4. As described in Chapter 4, the system was designed to work for both misuse and anomaly detection. Below are the details describing the flow of all paths in the system architecture.
Figure 5.4: The Proposed Intrusion Detection System Architecture
All the system calls invoked by the target container are being traced using system calls tracing tools such as Strace and Sysdig. After invoking all the system calls, the data cleaning component is responsible for reporting only the system calls names without any other values, such as arguments, return values, etc. A component follows the data cleaning that chooses which system system calls to use to construct the bag of system sequences based on a criteria described earlier in section 4.4. Then, the sequences construction component builds the bag of system sequences using epochs of the traced system calls. When the system starts running, the extracting features component extracts features from the normal profile. Several epochs files are needed to choose the unique normal sequences to build the features for the normal vector (details of the how many epochs are needed to build are described in Chapter 6). An additional feature to be added to that vector records the new sequences frequencies of the samples that were not among the constructed feature vector.

Hundreds of features usually needed to build the feature vector depends on the container operations and activities. Thus, a component was needed to be added to the architecture to perform dimensional reduction and feature selection. The need of that feature depends on users’ preferences, as discussed in detail in the discussion section 6.7. Two popular algorithms were used for that purpose: Principal component analysis (PCA) and The Correlation Feature Selection (CFS). The PCA was chosen due to its popularity for feature reduction and, the CFS was chosen because it was faster and gave promising results that improved the detection rate of the classifiers.

After building the feature vector, numerous normal epochs were used to
build the normal profile and store it in the database component in the system architecture to be used for the anomaly detection. Another database component stores most common attacks patterns’ sequences that are used against the targeted container, e.g. SQL, Apache, etc. to be used for misuse detection. The abnormal samples were built using the normal behavior feature vector. The system was able to detect unknown attacks, even it was misuse based system, as shown in Chapter 6.

The final components in the architecture are the classifiers components. For the anomaly detection, two algorithms were used to detect anomalies: the Interquartile range (IQR) and a K-nearest Neighbor that uses an euclidean distance-based metric. For the misuse detection, several machine learning algorithms were used to detect anomalies as follows: Artificial Neural Networks (ANNs), C4.5 decision tree, Random Forests, and Support Vector Machines (SVM).
Chapter 6

Implementations and Results

As mentioned in Chapter 4, training the classifiers should be modeled as quick as possible with minimum data, and maintaining high accuracy and efficiency. The reason for this is to obtain an efficient and accurate cloud intrusion system that works in real time which is one of the main requirements to build cloud intrusion detection systems.

After building the cloud environment, the MySQL container was used to gather the data during the running time in the cloud. Part of the normal activities were maximized in order to test the effectiveness of the proposed enriched representation used for both the simple anomaly detection systems and the sophisticated ones in the misuse detection systems.
6.1 Type of Experiments

Several experiments were conducted for multiple purposes, and not all the experiments’ results were presented here in this dissertation. Some were done for test purposes, some will be completed and published in separate research papers, and the experiments that fall into our research scope were presented here in this document. The types of experiments were as follows:

- Gathering normal and abnormal datasets from the MySQL container to test the proposed enriched representation.

- Gathering normal and abnormal datasets from MySQL and Apache containers running on the cloud to test the proposed anomaly and misuse host based intrusion detection systems. The MySQL container data was used for the rest of the research implementations to have a consistent dataset among all the proposed systems.

- Evaluating the effectiveness of the proposed distinct feature that was applied to the data representation and comparing the results to those results without applying this feature.

- Evaluating the effectiveness of implementing the Dimensional Reduction using the popular PCA algorithm. Also the CFS algorithm was preliminarily tested and showed promising results. The results of the CFS experiments and other experiments will be kept for the authors’ future work.

- n-Gram of Bag of System Calls (nBoSC) representation was preliminarily tested and it significantly reduced the size of used memory and stored data.
More experiments will follow in the future and present the results in separate research.

The main experiments that were done for this research have been discussed above. The ones that were preliminarily tested were out of this research’s scope and were left for future work. Many detail about the procedures of these main experiments were discussed in Chapters 4 and 5.

6.2 Classifiers Training

Before training the classifiers, the steps of the proposed systems must be deployed. After finishing the trace and data cleaning phases, these processes are also deployed in order as follows:

- Random 16 epochs (5000 system calls per epoch as in [24]) were needed to do the following$^1$:
  
  - Create the unique system calls list. 19 system calls were selected by applying the criteria mentioned in Chapter 4.
  
  - A window size of 10 was chosen to construct the sliding window of the sequences since it was used in [26] and [24] and proved to provide better results without dramatically affecting the efficiency of the classifiers.

---

$^1$Those sixteen epochs represent the normal profile that were used to derive the features from. The same method in [164] was used to build the profile. When reading the normal samples, the training stopped when first point on the curve slope was less than one new sequence per 10 traces.
Extracting the features by constructing the features vector where each bag of system calls sequence is considered as a feature. An additional feature (called the distinct feature) is added to the vector that determines that any sequence was not found in those sixteen epochs. The frequencies of the new sequences in the training sample would be the value of that feature corresponding to that specific sample. The total features of each vector was 2138 features including the distinct feature.

- A total of 1,080,000 system calls were used to form the normal and abnormal samples (695,000 normal and 385,000 abnormal) that were used by all the classifiers are discussed later in this chapter.

6.3 Cross-validation

To ensure that the training dataset was not leaking any information to the test dataset and to avoid overfitting problem, a 10-fold cross-validation technique (see Figure 6.1) was used to model classifiers to develop the intrusion detection systems to generalize the results to an independent dataset [45].

In k-fold cross-validation, the dataset $D$ is split randomly into $k$ exclusive subsets (folds) $D_1, D_2, ..., D_k$ are of about the same size. Each fold is trained and tested $k$ times, where each time $t \in 1, 2, ..., k$, it is trained on $D \setminus D_t$ and tested on $D_t$. [96]

In this research case, the original dataset is randomly partitioned into 10 equal sized subsamples. Each subsample is used one time as a validation sample
for testing and the remaining 9 subsamples were used for training. After covering all 10 subsamples, the average was taken to produce the estimation.

6.4 Classifiers Evaluation

All the classifiers used in this research were evaluated using well-known methods to evaluate the classifiers’ performance. They are based on the confusion matrix of two class problems having positive and negative class values, which in this case normal and abnormal classes. Table 6.1 shows the two class confusion matrix[36].

<table>
<thead>
<tr>
<th></th>
<th>Positive Prediction</th>
<th>Negative Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Positive Class</strong></td>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td><strong>Negative Class</strong></td>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>
Widely-used metrics were extracted from that matrix to measure the classifiers’ performance as follows [36]:

*The accuracy (Acc)* of the classifier is defined as

\[ Acc = \frac{TP + TN}{TP + FN + FP + TN} \]

*True Positive Rate (TPR)* is the percentage of the positive cases that were classified correctly as they belonged to the normal class, defined as follows:

\[ TPrate = \frac{TP}{TP + FN} \]

*False Positive Rate (FPR)* is the percentage of the negative cases that were misclassified as they belonged to the normal class, defined as follows:

\[ FPrate = \frac{FP}{FP + TN} \]

The above measures were used to evaluate the proposed intrusion detection system classifiers, with the goal of maximizing the true positive rate and minimizing the false positive rate as well as having the accuracy at a higher level. In addition, the performance of creating the model and producing the results is used for the misuse detection algorithms since it plays a major role when comparing the intelligent algorithms that were used in this research.
6.5 Anomaly Intrusion Detection Systems

Two anomaly intrusion systems were developed using common algorithms, applying the representation discussed in Chapter 4. These two algorithms, along with their performance evaluation, are described below.

6.5.1 Interquartile range (IQR) Classifier

The first algorithm used for anomaly detection to detect outliers was the IQR. As described in Chapter 2, the median (Q2) has to be found first to determine the quartiles. The first quartile (Q1) is the median of the lower half of the data where the third quartile (Q3) is the median of the upper half of the data [81]. It is calculated by finding the difference between the third quartile (Q3) and the first quartile (Q1) as follows [81]:

\[ IQR = Q3 - Q1 \]

6.5.1.1 Classifier Training

The IQR classifier was trained to recognize anomalies within the normal samples. The sample is considered to be an anomaly if it was below \( Q1 - (1.5)^*IQR \) or above \( Q3 + (1.5)^*IQR \).

The IQR was applied to 216 mixed samples (139 normal and 77 abnormal) to distinguish outliers. The output was evaluated using the criteria described in
section 6.4, as shown in Table 6.2

Table 6.2: Interquartile Range (IQR) Classifier Evaluation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interquartile Range (IQR)</td>
<td>92.13%</td>
<td>89.61%</td>
<td>6.47%</td>
</tr>
</tbody>
</table>

As mentioned in Chapter 2, one of the main drawbacks with the anomaly detection is the high false positive rate, as seen for the IQR classifier. Part of the reason was also due to the methods of generating the normal samples. As mentioned in the generation data section, some of the normal samples were intended to maximize the container usage within the normal usage in order to examine which classifier could reach the least FPR with low overhead performance. More details and comparisons with other classifiers are found in discussion section 6.7.

6.5.2 \textit{K}-nearest Neighbor Classifier

Modeling the data point using statistical distribution sometimes cannot be applied, especially if users do not have enough knowledge about the data distribution [138]. However, Knorr and Ng [95] proposed a distance-based approach, defined as: "A point in a data set is an outlier with respect to parameters $k$ and $d$ if no more than $k$ points in the data set are at a distance of $d$ or less from $P$" [138]. It does not require prior knowledge of data distribution as other statistical methods require.

The algorithm used to detect anomalies was based on a distance-based algorithm proposed by Ramaswamy et al. [138] that measures the distance of a point from its $k^{th}$ nearest neighbor. Each point was ranked on the basis of its
distance to its 10th nearest neighbor and declare the top n points in this ranking were declared to be anomalies.

6.5.2.1 Classifier Training

The classifier used for the detection system used an algorithm that computes $D_n^k$(Distance of point $P$ to its $k^{th}$ nearest neighbor) outliers (anomalies), as shown in Figure 6.2 [138]. Points would be added into an R* -tree index at the beginning of the for loop. Then, the R* -tree computes the $k^{th}$ nearest neighbor of each data point. $D^k$ stores the data points in increasing order and passes its value stored in $\text{minDkDist}$ to the function $\text{getKthNeighborDist}$ (Figure 6.3) [138].

Procedure computeOutliersIndex($k, n$)
begin
for each point $p$ in input data set do
    insertIntoIndex(Tree, $p$)
outHeap := $\emptyset$
minDkDist := 0
for each point $p$ in input data set do {
    getKthNeighborDist(Tree.Root, $p$, $k$, minDkDist)
    if ($p.DkDist > \text{minDkDist}$) {
        outHeap.insert($p$)
        if (outHeap.numPoints() > $n$) outHeap.deleteTop()
        if (outHeap.numPoints() = $n$)
            minDkDist := outHeap.top().DkDist
    }
}
return outHeap
end

Figure 6.2: Index-Based Algorithm for Computing Anomalies

When calling the function $\text{getKthNeighborDist}$, it computes $D^k(p)$ for data point $p$ by examining the nodes in the R* -tree using linked list $\text{nodeList}$. It
sorts the linked elements in ascending order of their minimum distance \((MIDIST)\) from \(p\). The practical details can be found in [138]

**Procedure** getKthNeighborDist(Root, \(p\), \(k\), \(minDkDist\))

begin
    nodeList := \{ Root \}
    \(p.Dkdist := \infty\)
    nearHeap := \(\emptyset\)
    while nodeList is not empty do {
        delete the first element, Node, from nodeList
        if (Node is a leaf) {
            for each point \(q\) in Node do
                if \((dist(p, q) < p.DkDist)\) {
                    nearHeap.insert(\(q\))
                    if (nearHeap.numPoints() > \(k\)) nearHeap.deleteTop()
                    if (nearHeap.numPoints() = \(k\))
                        \(p.DkDist := dist(p, nearHeap.top())\)
                    if \((p.Dkdist \leq minDkDist)\) return
                }
        }
        else {
            append Nodes children to nodeList
            sort nodeList by \(MINDIST\)
        }
    }
    for each Node in nodeList do
        if \((p.DkDist \leq MINDIST(p,Node))\) delete Node from nodeList
end

Figure 6.3: Computation of Distance for 10\(^{th}\) Nearest Neighbor

The Euclidean Squared Distance Metric (shown below) was used as the distance measure because it produces fewer and less expensive computations.

\[
d = \sum_{i=1}^{n} (x_i - y_i)^2
\]
The above algorithms applied to the examined data and the results are shown in Table 6.3

Table 6.3: \(K\)-nearest Neighbor Classifier Evaluation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(K)-nearest Neighbor</td>
<td>87.04%</td>
<td>81.82%</td>
<td>10.07%</td>
</tr>
</tbody>
</table>

Also the results were expected because of the nature of generating the normal data where some of it was to maximize the container usage within normal ranges. When applying more sophisticated machine learning algorithms as in the following sections, the results were improved significantly because they considered more factors and features to differentiate between anomalies and normal data. Further discussion and analysis can found in the discussion section 6.7.

6.6 Misuse Intrusion Detection Systems

Two anomaly algorithms have been discussed in the previous sections where the results of the accuracy, FPR, and TPR, were relatively low due to the natural use of those algorithms that detect outliers. In this section, more sophisticated machine learning algorithms were introduced to show how they were used to detect intrusions.

Besides the capabilities of the machine learning algorithms used, in terms of providing good classification, the enriched representation helped some of the algorithms to achieve a 100% accuracy detection rate, as shown in the following sections. As described in Chapter 2, the misuse detection can only detect known
attacks that were found in the predefined database. In this dissertation, the misuse
detection could also detect unknown attacks that were not in the defined attacks
patterns because the patterns of the attacks were formed using the proposed En-
riched Representation mechanism, as explained in Chapter 4. The four machine
learning algorithms used are described below.

6.6.1 Artificial Neural Networks (ANNs) Classifier

For the Neural Network algorithms, as described in Chapter 2, The Multilayer
Perceptron algorithm was used because it was proven to be a very effective alter-
native for most of the statistical techniques [142]. As Multilayer Perceptron was
described in Chapter 2, the training algorithm used is called backpropagation, and
its computation is straightforward [141], as described in the Classifier Training
section 6.6.1.1.

6.6.1.1 Classifier Training

The algorithm completes the provided training samples \((x(n), d(n))\) as follows
[74]:

1. Initializing, with the assumption that no prior knowledge is available to the
classifier. The weights and thresholds are picked from uniform distributions
where the mean is zero. Also, the determined variance is set to make the
standard deviation of the induced local fields of neurons at the transition
between linear and standards parts of the sigmoid activation function.

87
2. *Presentations of Training Examples*, where the network is presented as epochs, denoted by \((\mathbf{x}(n), \mathbf{d}(n), \text{input vector } \mathbf{x}(n))\), in order to compute the Forward and Backward computation described in steps 3 and 4.

3. *Forward Computation* allows the training example in the input to be applied to the input layer of sensory nodes and the desired response vector \(\mathbf{d}(n)\) presented to the output layer of computation nodes. Computing the induced local fields and function signals of the network proceeds forward layer-by-layer through the network. When using the sigmoid function, the output signal of neuron \(j\) in layer \(l\) is computed using this formula:

\[
y^l_j = \varphi_j(\nu^l_j(n))
\]

For neuron \(j\) at the first hidden layer (\(l=1\)):

\[
y^{(0)}_j = x_j(n)
\]

\(x_j(n)\) is the \(j^{th}\) element of the input vector \(\mathbf{x}(n)\). If the output layer contains neuron \(j\), set \(y^L_j = o_j(n)\), where \(l=L\) and \(L\) is the depth of the network. Thus, the error signal is computed using:

\[
e_j(n) = d_j(n) - o_j(n)
\]

where \(d_j(n)\) is the \(j^{th}\) element of the desired response vector \(\mathbf{d}(n)\).

4. *Backward Computation* occurs when the local gradients of the network are computed. Also, the synaptic weights of the network in layer \(l\) are adjusted.
5. *Iteration* occurs when the new training epochs computation takes place by iterating the forward and backward computations until the defined threshold is reached.

The PCA algorithm was also applied to the Multilayer Perceptron algorithm to reduce the features used. The PCA produced 36 attributes from the dataset which were used as inputs for the Multilayer Perceptron algorithm and the classes (normal and abnormal) used as outputs. One hidden layer was used with 18 nodes. The results of the Multilayer Perceptron algorithm classifier are presented in Table 6.4.

Table 6.4: Artificial Neural Networks (ANNs) Classifier Evaluation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Artificial Neural Networks (ANNs)</td>
<td>96.97%</td>
<td>97.0%</td>
<td>1.3%</td>
<td>13.28 seconds</td>
</tr>
</tbody>
</table>

The Artificial Neural Networks (ANNs) algorithms run slowly due to their nature of working with multilayers, so there was a need to apply the dimensional reduction for the hundred produced features. Other algorithms (presented later in this chapter) run fast without the dimensional reduction, and they run even faster when applying the reduction. Detailed comparisons between the performance of those algorithms with/without applying the reduction is presented in this chapter as well.
6.6.2 C4.5 Decision Tree Classifier

Chapter 2 has shown that the C4.5 algorithm is the most well-known algorithm for building a decision tree. This section describes how the learning process occurs and how the classifier is trained. Then, evaluation results are shown. Training the C4.5 classifier is explained in many research papers as in [137] and is summarized by many others, such as in [140].

6.6.2.1 Classifier Training

It was discussed in Chapter 2 how the decision trees are different in terms of how each algorithm constructs its tree from the training set. In this research, the C4.5 decision tree algorithm is chosen as one option for the misuse intrusion detection. Ruggieri in [140] summarized the pseudocode of how C4.5 constructs its tree as follows:

Let $T$ be the set of cases dedicated to the node where each case determines the attributes and class' values. $C$ is the value of the class; the tree construction follows these steps to complete building the decision tree:

FormTree(T)
(1) ComputeClassFrequency(T);
(2) If OneClass or FewCases
   return a leaf;
   Create a decision node N;
(3) For Each Attribute A
ComputeGain(A);

(4) N.test = AttributeWithBestGain;

(5) if N.test is continuous
    find Threshold;

(6) For Each T' in the splitting of T
(7) if T' is Empty
    Child of N is a leaf
else
    Child of N = FormTree(T');

(9) ComputeErrors of N;
    return N

In step 1, the weighted frequency of the $C$ for the cases in $T$ is computed. Step two determines whether all cases belong to the same class $C_j$ or if the frequency of the cases is less than a defined threshold, and then the node is assigned as a leaf. In step 3, the information gain is calculated if $T$ contains cases belonging to 2 or more classes. The attribute with the greatest information gain is selected for the test at the node (N) in step 4. In step 5, the threshold is computed to be the highest value of the training set if the selected attribute was continuous. In step 6, a decision node has children ($s$) if $T_1, ..., T_s$ are the chosen sets for the splitting. The child node is set to be a leaf if $T_i$ is empty; Otherwise, in step 8, the same operation is recursively applied to the set which contains $T_i$ and cases with unknown values. Lastly, in step 9, the node’s classification error is computed to be the sum of the error of the child nodes’ errors. If it is greater than classifying all the cases as belonging to the most frequent class, then the node is a leaf and
all other subtrees are deleted [140].

The **Information Gain** is used by the C4.5 algorithm to reduce a bias towards multi-valued attributes. The information gain of attribute \( a \) for a set of cases \( T \) was computed as follows [140]:

\[
gain = info(T) - \sum_{i=1}^{s} \frac{|T_i|}{|T|} \times info(T_i)
\]

The expected information that is needed to classify a sample of the dataset is obtained by computing the entropy function [86] as follows [140]:

\[
info(T) = - \sum_{j=1}^{N\text{Class}} \frac{freq(C_j, T)}{|T|} \times \log_2 \left( \frac{freq(C_j, T)}{|T|} \right)
\]

The splitting of subsets of data, which is the ratio of information gain to its split information, can be computed as follows: [140]:

\[
split(T) = - \sum_{i=1}^{s} \frac{|T_i|}{|T|} \times \log_2 \left( \frac{|T_i|}{|T|} \right)
\]

After applying all these computations to train the C4.5 algorithm classifier to make predictions on the test dataset, the results are presented in Table 6.5.
Table 6.5: C4.5 Classifier Evaluation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5 Decision Tree</td>
<td>98.99%</td>
<td>99.0%</td>
<td>1.4%</td>
<td>1.87 second</td>
</tr>
</tbody>
</table>

The C4.5 algorithm’s predictions were better than the same data applied to the Multilayer Perceptron algorithm with 1.4% FPR. In the following sections, the same dataset also is analyzed by two other different algorithms.

6.6.3 Random Forests Classifier

A more complex version of the decision tree (discussed in the previous section) is the Random Forest (RF) algorithm; its efficiency and advantages were discussed in Chapter 2. Although the improvement in the results of the misuse detection has been shown by the decision tree C4.5 algorithm, the results of the Random Forest shows even better performance and higher accuracy. RF constructs a multitude of decision trees at training time, as explained in the next section, and the results of this process are shown in the table results in section 6.6.

6.6.3.1 Training the Classifier

The Random Forest classifier was trained using the process described by its creator, Breiman, in [42] and was summarized in [113] as described in the following steps:
• The classifier starts by drawing \( n_{\text{tree}}^2 \) bootstrap samples \( X_i \) which is derived from the original dataset.

• The Unpruned classification technique\(^3\) is grown for each bootstrap sample. At each node, the best split will be assigned from the randomly sampled \( m_{\text{try}}^4 \) of the predictors and not among all of them.

• New data are predicted by aggregating the predictions and taking the majority votes of the \( n_{\text{tree}} \) trees.

The error rate of predicting the data that were not in the bootstrap samples (out-of-bag (OOB)) \( ERR_{\text{OOB}} \) were estimated as follows:

1. Predicting the out-of-bag elements at each bootstrap iteration by using the tree that was grown using the bootstrap samples.

2. Aggregating the out-of-bag predictions \( (g_{\text{OOB}}) \). For the \( i^{\text{th}} \) element \( (y_i) \) of the training data is considered in which the \( i^{\text{th}} \) is out-of-bag. On average, each data point of the samples would be OOB 36% of the iterations; thus, the predictions would be aggregated. The error rate \( (ERR_{\text{OOB}}) \) was estimated using this formula [161]:

\[ \text{ERR}_{\text{OOB}} \]

\(^2\)The number of the bootstrap samples

\(^3\)Random Forests performs the Pruning technique which reduces the size of the learning tree without reducing accuracy.

\(^4\)The number of different predictors at each node [161].
\[
ERR_{OOB} = \left(\frac{1}{n_{tree}}\right) \sum_{i=1}^{n_{tree}} [y_i - g_{OOB}(X_i)]^2
\]

The Random Forest classifier was trained using the same data used by other algorithms, described above with 100 trees and 10 cross-validation folds. The results are shown in Table 6.6.

Table 6.6: Random Forests Classifier Evaluation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random Forests</td>
<td>100%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>3.52 seconds</td>
</tr>
</tbody>
</table>

As shown in the result, the Random Forest classifier could reach a 100% accuracy and 0.00% FPR using the proposed data representation mechanism; further details are provided in section 6.7.

6.6.4 Support Vector Machines (SVM) Classifier

The Sequential minimal optimization (SMO) algorithm was used to train the Support Vector Machines classifier. An overview of how the SMO algorithm was used to train the classifier in order to predict class labels for the test dataset is given below. The same data that are used by other algorithms were used here in order to provide realistic comparisons as discussed in section 6.7. Other research and published papers provided the resources that explained in detail how this algorithm works, and below is an overview that was derived from [52], the original paper by John Platt [135], and from the co-inventor, Vladimir Vapnik, of the SVM
6.6.4.1 Classifier Training

The classifier used the SMO algorithm to train the support vector machines. The process of how the process took place, which was derived from [135] and summarized by [52], is described below. A detailed pseudocode follows the description of the classifier training steps. Before writing the details of how the SMO trains the SVM classifier, Kun et al. [56] simplified the SMO algorithm by a pseudocode, as follows:

1. Loop through the training set until reaching no possible improvement
2. Select two multipliers, \( a_1 \) and \( a_2 \), by using heuristics
3. Optimize by assuming all other multipliers are constant
4. End Loop

The support vector machine computes a linear classifier where the SVM is the hyperplane that separates positive samples from the negative ones with a maximum margin, as described in Chapter 2. The output of a linear SVM is produced by this function:

\[
 f(x) = w^T x + b
\]

Where \( w \) is the normal vector to the hyperplane and \( x \) is the input vector.
The hyperplane separating the normal points from the abnormal is \( f(x) = 0 \), and the nearest points that lie on the planes are \( f(x) = \pm 1 \). The prediction of the class normal samples will be \( f(x) \geq 0 \), and the abnormal samples will be predicted if \( f(x) < 0 \).

There are three main steps the SMO algorithm goes through: selecting \( \alpha \) parameters, optimizing these parameters, and then computing the threshold \( b \). The simplified version of the SMO is presented below to give an overview of how the algorithm works, as summarized in [52].

**Input:**

\[ C: \text{ regularization parameter} \]

\[ \text{tolerance: numerical tolerance} \]

\[ \text{maxPasses: maximum number of times to iterate over } \alpha \text{'s within the training data} \]

**Output:**

\[ \alpha \in \mathbb{R}^n: \text{ Lagrange Multipliers for the solution} \]

\[ b \in \mathbb{R}: \text{ solution’s threshold} \]

Initialize \( \alpha_i = 0, \forall i, b = 0 \)

Initialize \( \text{passes} = 0 \)

**while ( passes < maxPasses)**

\[ \text{numChangedAlphas} = 0 \]

**for** \( i = 1, ..., m \),
calculate $E_i = f(x^{(i)}) - y^{(i)}$; See 5

if $((y^{(i)}E_i < -\text{tolerance} \&\& \alpha_i < C) \mid (y^{(i)}E_i > \text{tolerance} \&\& \alpha_i > 0))$

select $j \neq i$ randomly

calculate $E_i = f(x^{(i)}) - y^{(i)}$

save old $\alpha'$s: $\alpha^{(old)}_i = \alpha_i$, $\alpha^{(old)}_j = \alpha_j$; See 6

compute $L$ and $H$; See 7

if $(L == H)$

continue to next $i$

compute $\eta$; See 8

if $(\eta >= 0)$

continue to next $i$

compute and clip new value for $\alpha_j$

if $(||\alpha_j - \alpha^{(old)}_j|| < 10^{-5})$

continue to next $i$

determine value for $\alpha_i$

compute thresholds $(b1$ and $b2)$

compute $b$

5It computes a kernel function which measures the similarity between the input vector and the trained vectors; where the $E$ is the error between the SVM output on the $k^{th}$ sample and the true label

6 $\alpha^{(old)}_i$ is the old value of $\alpha_i$ before optimization

7finding bounds $L$ and $H$ where $L \leq \alpha_j \leq H$ must hold in order for $\alpha_j$ to satisfy the constraint that $0 \leq \alpha_j \leq C$

8Computing the $\eta$ parameter in order to perform the calculations in later steps to maximize the objective function. If the value becomes outside the determined bounds $L$ and $H$, the value of $\alpha_j$ will be clipped in order to lie within that range, as shown in the calculations below this step
numChangedAlphas = numChangedAlphas + 1

end if

end for

if (numChangedAlphas == 0)
    passes = passes + 1
else
    passes = 0
end while

The results from training the classifier using the SMO algorithm to train the support vector machines with the same dataset used previously are shown in Table 6.7.

Table 6.7: SVM Classifier Evaluation

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>100%</td>
<td>100.00%</td>
<td>0.00%</td>
<td>1.45 second</td>
</tr>
</tbody>
</table>

The above results are from the full dimension dataset. In the discussion section 6.7 additional results are shown for the PCA algorithm when it was applied for dimensional reduction.

6.7 Discussion

There are requirements for designing a cloud based intrusion detection systems that were discussed in Chapter 2. One of the main requirements is that the system must work in real time. In order to design an intrusion detection system that
works in real time, the detection has to be performed immediately and relatively fast.

The proposed Enriched Representation mechanism was designed to improve the traditional intrusion detection systems in terms of the detection rate, accuracy, learning process performance, building the normal profile, and the normal & abnormal samples with minimum amount of data, and detecting unknown attacks by misuse detection. All the above mentioned goals were achieved, as shown in the results sections and discussed here in this section.

Building an initial normal profile was needed in order to use its sequences as features to form the normal and abnormal samples’ feature vectors. Several experiments were done to achieve the minimum data needed to form the normal profile of a container running in cloud environment. Only 19 unique system calls were needed to form the unique sequences of the normal profile because there was a strict criteria (discussed in detail in Chapter 4) that was followed to choose which system calls could be useful to detect anomalies. The profile was built using only 16 epochs (5,000 system calls each) which is a relatively small amount of data compared to other systems discussed in the literature review chapter (Chapter 3).

After building the normal profile, its features were extracted (in addition to the distinct feature that was discussed in section 6.7.1) to form the normal and abnormal samples that were used to train and test the classifiers. The used features were used to construct the feature vector for both normal and abnormal samples. Only the sequences found in the normal profile were used as features, which helped the classifiers to achieve 100% accuracy and detection rate. The normal samples registered relatively similar sequences counts of the features. On the other hand,
the abnormal samples registered a significantly different count of the registered features. Many of the features registered zero count in the abnormal samples and the attacks patterns. That was because the attacks used different sequences that were not registered in the features vector.

6.7.1 The Distinct Feature Evaluation

An additional feature was added to the features vector that registered unique sequences that were not seen in that specific epoch. Usually the normal samples register very few unseen sequences that were not found in the normal profile, unlike the attack patterns which register very high counts of unseen sequences at that feature. Table 6.8 shows how this feature improved the accuracy and the detection rate of the C4.5 decision tree algorithm as an example.

<table>
<thead>
<tr>
<th>The additional feature</th>
<th>Accuracy</th>
<th>Detection rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not included in the feature vector</td>
<td>97.01%</td>
<td>97.0%</td>
</tr>
<tr>
<td>Included in the feature vector</td>
<td>98.99%</td>
<td>99.0%</td>
</tr>
</tbody>
</table>

In addition to the improvement that the Enriched Representation achieved, there were additional improvements, explained in the next section 7.1, that helped to achieve process performance and storage efficiency. Those improvement methods were preliminarily tested, and the results are not presented here because it is out of the research scope and will be presented in separate research paper.
Table 6.9: Classifiers Evaluation when Dimensional Reduction is Applied

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5 Decision Tree</td>
<td>97.98%</td>
<td>98.00%</td>
<td>2.8%</td>
<td>0.14 second</td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>97.69%</td>
<td>97.7%</td>
<td>2.4%</td>
<td>0.47 second</td>
</tr>
<tr>
<td>Random Forests</td>
<td>99.49%</td>
<td>99.5%</td>
<td>2.0%</td>
<td>0.62 second</td>
</tr>
<tr>
<td>Artificial Neural Networks (ANNs)</td>
<td>96.97%</td>
<td>97.00%</td>
<td>1.3%</td>
<td>13.28 seconds</td>
</tr>
</tbody>
</table>

6.7.2 Dimensional Reduction Evaluation

Since constructing the feature vectors of the samples were based on the sequences’ normal profile, the vectors ultimately came up with hundreds of features that identify the sample, whether it was normal or abnormal. The classifiers were very successful in building the model very fast, even with the hundreds of features; however, when applying the dimensional reduction performed even faster.

The Principal Component Analysis (PCA) algorithm used for the dimensional reduction has been used for a long time for that purpose with proof of effectiveness. Applying the PCA to the data had the side effect\(^9\) of reducing accuracy, as shown in Table 6.9, compared to the results without reducing the dimensions where all the features were used to predict intrusions, as shown in Table 6.10.

Without applying the dimensional reduction feature, the classifiers consumed time in order to process the algorithm, but it took less time than applying the machine learning algorithms on the data without reducing the dimensions. The feature selection, instead, could make better performance. The CFS algorithm was

---

\(^9\)There was a small percentage of a trade off between accuracy and performance efficiency, and thus, this option of the dimensional reduction system feature is left optional to users to enable it in the system or not
Table 6.10: Classifiers Evaluation without Applying Dimensional Reduction

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Accuracy</th>
<th>True Positive Rate (TPR)</th>
<th>False Positive Rate (FPR)</th>
<th>Process Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C4.5 Decision Tree</td>
<td>98.99%</td>
<td>99.0%</td>
<td>1.4%</td>
<td>1.87 seconds</td>
</tr>
<tr>
<td>Support Vector Machine (SVM)</td>
<td>100%</td>
<td>100%</td>
<td>0.0%</td>
<td>1.45 seconds</td>
</tr>
<tr>
<td>Random Forests</td>
<td>100%</td>
<td>100%</td>
<td>0.0%</td>
<td>3.52 seconds</td>
</tr>
</tbody>
</table>

tested on a small set of data and provided promising results and performed better than the PCA algorithm. This needs further study and will be left for future work.

The classifiers performed faster when applying the PCA algorithm to reduce the dimensions as well as using less memory and storage since the features were reduced significantly. However, the process time to perform the PCA algorithm was 27.56 seconds which is very long time that makes the dimensional reduction feature (PCA in this case) was not preferable. The artificial network classifier (ANN) could not perform properly with the accepted running time without reducing the dimension of the feature vector. Another example is the decision tree classifier, which was trained in about 1.4 seconds without dimensional reduction and with an accuracy of 98.99%; on the other hand it took it around 0.14 second to be trained when reducing the number of the used features, but with lower accuracy of 97.98%. It is a small difference in terms of performance but it becomes huge when processing ten of thousands of samples. As described earlier, it is a trade off between performance and accuracy which depends on users’ and enterprises’ preferences and the nature of applications.

The proposed intrusion systems were designed in a way that their classifiers learned the models relatively quickly with the least amount of data. Many experiments have been done in order to determine the minimum possible amount of data needed to train and test the models. Some of the built classifier models needed around 0.47 second to build the model and produce the results, such as the
SVM classifier, whereas the Neural Network model needed around 13.28 seconds due to its nature of complexity to build the model.

Building the samples was relatively fast, with the average performance time for creating the normal sample being 1.057 seconds using a typical machine (64 bit with average hardware setup). The abnormal samples’ performance time average was 1.648 seconds due to the high frequencies of unknown sequences that were not in the normal profile sequences.

The misuse detection systems performed much better than the anomaly detection systems in terms of accuracy and detection rates, as well as Low False Positive rates. The reasons for that significant difference in the results were the methods used in both systems, as well as the way the normal data were generated. Regarding the detection methods, the misuse detection used sophisticated machine learning algorithms that used all the features together to form the classifiers models. Having the normal profile sequences as features helped the classifiers to detect anomalies. Moreover, the representation of the feature vector helped the classifiers to detect unknown attacks that were not in the attacks patterns database. Some attacks (e.g. port scanning attack) were detected even they were not in the database of the attacks patterns, and that was because the attacks patterns formed using the normal profile features vector. Regarding the generation of normal data, when the container is one ordinarily used by manual and automated normal activities, some of the operations (e.g. `select` queries) were maximized to the maximum level that is consider normal\(^\text{10}\) which was because it was necessary

\(^{10}\)The threshold of that level was chosen by experiments
to generate normal data that could be considered as anomalies by weak classifiers as in the anomalies detection case (e.g. IQR classifier). The misuse detection classifiers were able to classify those samples as normal as they were intended to be.
Chapter 7

Conclusion

Intrusion detection systems were proposed for containers running in a cloud environment. The motivation for this research was the discovery after a review of recent research, showed that there is a lack of research regarding the cloud based intrusion detection systems specifically for containers. Chapter 2 describes the requirements of cloud based intrusion detection systems. The applicable requirements were applied to the proposed systems and were able to detect attacks very quickly and accurately. The system performed in real-time maintaining maximum True Positive Rates and minimum False Positive Rates.

A cloud environment was built in the security labs at the Florida Institute of Technology. The design of that environment along with the assigned components’ IPs are discussed in Chapter 5, which also describes the data source used in this research, which were system calls. They were used because they were proved to be a preferred source of data since they are able to access the core kernel functions
The system calls that were generated by targeted containers were traced and pre-processed. The pre-processing procedure included selecting a subset of system calls determined by a criteria discussed in detail in Chapter 4. Only this subset of system calls was chosen to ensure that creating the normal behavior profile, and the normal & abnormal samples occurs very quickly in order to have a real-time detection. Thus, an Enriched Data Representation mechanism was proposed and implemented for that purpose. The mechanism used different models to create the feature vectors used by the intrusion detection classifiers.

The filtered system calls were split into epochs where each epoch contained approximately the same amount of system calls. A few numbers of epochs were used to choose the subset of the used system calls. Other epochs were used to generate the normal profile feature vector that was used to generate the normal and abnormal samples by mapping their features to the normal profile feature vector.

The sliding window technique was used to produce sequences to be used to build a normal profile feature vector that represented the monitored container. The bag of system calls representation [84] was used for the generated sequences. Each sequence was used as a feature for the feature vector that represented the normal profile. More details are discussed in Chapter 4.

In order to test the proposed intrusion detection systems, normal and abnormal samples were generated. The normal operations performed both manual and automated activities on the container to maximize the chance of imitating the real world activities and data. The same procedure was performed with the abnormal samples where real attacks were used in order to generate abnormal
samples and patterns that were used later for training the classifiers.

The proposed framework for both anomaly and misuse intrusion detection is discussed in Chapter 5. It shows how the data flows in the system for both detection techniques. The architecture of the framework starts by tracing the container generated system call followed by cleaning the data phase. After that, the unique system call list is generated to feed the sequences construction component. The features are extracted after that, and an additional distinct feature is added to them. The dimensional reduction process then takes place. Next, the normal profile is created and updated regularly, and the same is true for the attacks patterns. The tested data follows their path to the classifier and are assigned to either normal or abnormal classes.

As stated in Chapter 6, 16 random epochs were needed to create the unique system calls list and create the normal behavior profile to be used to extract the features for both normal and abnormal samples. A total of 1,080,000 system calls were used to form all samples.

Statistical, instance, and classification based techniques for anomaly and misuse detection were used in this research. For the statistical-based, the interquartile range (IQR) was used to detect outliers. For the instance-based, the K-nearest neighbor classifier was used to detect outliers as well. For the classification-based detection, more sophisticated classification algorithms were used as follows: artificial neural networks (ANNs), C4.5 decision tree, random forests (RF), and support vector machines (SVM). The results show that using the machine learning techniques give more accurate detection and maintain high true positive rates and low false positive rates, where some algorithms, such as SVM and RF, achieved a 100%
detection rate with a 0.00% false positive rate.

Detailed steps in Chapter 6 show how the classifiers were trained and tested. Also, more detail is given on the classifiers’ performances and how they compared to other classifiers. In addition, the additional distinct feature and the dimensional reduction techniques’ performances were evaluated and discussed in the same chapter.

7.1 Future Work

The proposed system can be improved by enriching the data representation even more in order to build the model even faster. Lei et al. [109] proposed SPEAKER, which is a container security mechanism that “dramatically reduces the number of available system calls to a given application container by customizing and differentiating its necessary system calls at two different execution phases, namely, booting phase and running phase.” SPEAKER is a complementary mechanism to this dissertation’s data representation mechanism which will improve the selection of the system calls used for constructing the features vector.

*n-Gram of Bag of System Calls Sequences*  

In this research, a new concept called *n-Gram of Bag of System Calls Sequences* was introduced and preliminarily tested. Unlike servers, hosts, or virtual machines, which have difficulty sustaining consistent activities due to their complex activities; the application container usually has consistent activities where the designing sequence of sequences is valid since a proof of concept was conducted with a pre-
liminarily test that showed promising results\(^1\). It reduced the size of the database of the normal profile and the normal/abnormal samples significantly, which saved space on disk and memory as well as reducing the amount of time and processing to detect anomalies.

Also, the system can be improved when the following points to overcome its limitations are considered:

- Adding some of the traced system calls arguments that is concerned with the running application.
- The system was specified for the containers during the run time. It could include monitoring them while building the container, as well ensuring there are no attacks sneak during booting.
- The proposed system assumes that the container and its host have no attacks. However, it can be modified to be capable of detecting an existing attack, whether on the container itself or on its host.
- The current system only detects attacks, without specifying what type of attack, but it can be improved to provide that important detail in order to devise ways to prevent them.

\(^{1}\)The results of more thorough tests will be published in a different research paper
Bibliography


[22] Iso/iec 27000. 2016.


[76] Victoria Hodge and Jim Austin. A survey of outlier detection methodologies. 

[77] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection 
using sequences of system calls. *Journal of computer security*, 6(3):151–180, 
1998.

[78] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong- 
system based on hierarchical clustering and support vector machines. *Expert 

[79] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedfor- 
ward networks are universal approximators. *Neural networks*, 2(5):359–366, 
1989.

[80] Wenjie Hu, Yihua Liao, and V Rao Vemuri. Robust support vector machines 

[81] Barbara Illowsky and Susan L Dean. *Introductory statistics*. OpenStax Col- 
lege, Rice University, 2013.

[82] Keith R Jackson, Lavanya Ramakrishnan, Krishna Muriki, Shane Canon, 
Shreyas Cholia, John Shalf, Harvey J Wasserman, and Nicholas J Wright. Performance analysis of high performance computing applications on the amazon 
web services cloud. In *Cloud Computing Technology and Science (CloudCom), 


detection in cloud systems. In *Information Technology: New Generations*

tional Intelligence, Communication Systems and Networks (CICSyN), 2013

[92] Dong Seong Kim and Jong Sou Park. Network-based intrusion detection
with support vector machines. In *International Conference on Information

[93] Edwin M Knorr and Raymond T Ng. Finding intensional knowledge of

[94] Edwin M Knorr, Raymond T Ng, and Vladimir Tucakov. Distance-based
outliers: algorithms and applications. *The VLDB Journal The International

[95] Edwin M Knox and Raymond T Ng. Algorithms for mining distance-based
outliers in large datasets. In *Proceedings of the International Conference on

[96] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy
estimation and model selection. In *IJcai*, volume 14, pages 1137–1145. Montreal,

[97] Andrew P Kosoresow and SA Hofmeyer. Intrusion detection via system call


