



CHALMERS
UNIVERSITY OF TECHNOLOGY



UNIVERSITY OF GOTHENBURG

Refining Security Monitoring Techniques for Container-Based Virtualisation Environments

Master's thesis in Computer science and engineering

MARCUS LINDVÄRN
ZACK LUNDQVIST

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

MASTER'S THESIS 2021

Refining Security Monitoring Techniques for Container-Based Virtualisation Environments

MARCUS LINDVÄRN
ZACK LUNDQVIST



UNIVERSITY OF
GOTHENBURG



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Refining Security Monitoring Techniques for Container-Based Virtualisation Environments

Marcus Lindv rn & Zack Lundqvist

  MARCUS LINDV RN, 2021.

  ZACK LUNDQVIST, 2021.

Supervisor: Rodi Jolak, Software Engineering Department

Examiner: Christian Berger, Software Engineering Department

Master's Thesis 2021

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

SE-412 96 Gothenburg

Telephone +46 31 772 1000

Typeset in L T X

Gothenburg, Sweden 2021

MARCUS LINDVÄRN

ZACK LUNDQVIST

Department of Computer Science and Engineering

Chalmers University of Technology and University of Gothenburg

Abstract

Context: Virtualisation is a vital part of many industries' software deployment. When virtualisation became popular, it was more or less synonymous with virtual machines and hypervisors. Since then, a newer form of virtualisation has surged in popularity, containers. Containers provide improvements over traditional hypervisors in several aspects, with lower overhead and short boot and shutdown times often being referenced.

Problem: However, due to the way containers operate, they do not achieve the same level of isolation, an essential attribute in security. Containers share kernel with the host and other containers running on the host. A shared kernel means the attack surface differs from hypervisors, causing an elevated need for proper monitoring and investigation of potential monitoring techniques for detecting attacks, threats or misbehaving containers.

Objective: This study aims to understand what container monitoring techniques are available and how they operate. Moreover, it explores novel container monitoring techniques providing better efficiency and coverage of the STRIDE threat model.

Approach: The first objective is realised by conducting a literature review using the snowballing approach. The second objective is realised by following the design science research methodology.

Results: As a result, a container monitoring technique is created and refined over four iterations. This technique uses the Isolation Forest algorithm to detect anomalies in system call traces. The Isolation Forest algorithm enables unsupervised anomaly detection while providing multiple advantageous characteristics in terms of efficiency and detection.

Evaluation: In order to evaluate and compare the proposed monitoring technique with other techniques, a framework is developed to support the use of different anomaly detection and feature extraction algorithms, streamlining the evaluation process.

Conclusion: The resulting technique detects all attacks included in the evaluation while keeping an average FPR below 3%.

Keywords: Computer, science, computer science, engineering, project, thesis, security, container, monitoring, anomaly detection.

Acknowledgements

We want to say a big thank you to our supervisor Rodi Jolak for all the guidance and helpful discussions he has provided throughout the project. He has given us recommendations and new perspectives that have enabled us to progress past the various roadblocks we have encountered, no matter their nature or immensity. His feedback and mentorship have also kept us on the right track and helped us to focus on the task at hand. We would also like to thank our examiner Christian Berger, who has helped us avoid pitfalls and polish the thesis to its detailed and concise state by giving constructive feedback from a critical point of view.

Marcus Lindvörn & Zack Lundqvist, Gothenburg, June 2021

Contents

List of Figures	xiii
List of Tables	xv
1 Introduction	1
1.1 Problem Domain and Motivation	2
1.2 Research Questions and Research Goals	3
1.3 Contributions	3
1.4 Scope	4
1.5 Structure of the Thesis	4
2 Background	5
2.1 Virtualisation	5
2.1.1 Virtual Machines	6
2.1.2 Containerisation	6
2.1.3 Firecracker	7
2.2 Docker	8
2.3 Threat Landscape and Security	8
2.3.1 STRIDE	9
2.4 Threat Protection Approaches	9
2.4.1 Static Analysis	10
2.4.2 Behavioural Analysis	10
2.4.2.1 Machine Learning-based Analysis	10
2.4.3 Runtime Security	11
2.5 Related Work	11
3 Approach	15
3.1 Snowballing Search Approach	15
3.1.1 Tools and Traceability	15
3.1.2 Search Strategy	16
3.1.3 Study Selection Criteria	17
3.1.4 Study Selection Procedure	17
3.1.5 Data Extraction and Synthesis	17
3.2 Refining a Monitoring Technique	18
4 Results of Literature Review	21
4.1 Snowballing	21

4.1.1	Extracted Data	21
4.1.2	Citation Matrix	30
4.2	Inter-rater Reliability Test	31
4.3	Literature Review Analysis	32
5	Implementation	35
5.1	Data Source	35
5.1.1	Alternative Sources	36
5.2	Data Representation	37
5.2.1	Bag of System Calls	37
5.2.2	One-hot Encoding	38
5.3	Feature Extraction	38
5.3.1	Frequency Vector	39
5.3.2	Sliding Window	40
5.3.3	n-gram	41
5.4	Anomaly Detection	42
5.4.1	Baseline Technique: k-nearest neighbours	42
5.4.2	Refined Technique: Isolation forest	43
5.5	Simulation Environment Tools	44
5.5.1	Heimdall - System Call Monitoring Tool	44
5.5.2	Hlin - Anomaly Detector Evaluation Framework	46
6	Evaluation	49
6.1	Simulation Environment	49
6.1.1	Creating Vulnerable Docker Containers	50
6.1.2	Collecting System Call Logs	51
6.1.3	Normal Load Generation	51
6.1.4	Exploitation	51
6.2	Evaluation Metrics	52
6.3	Iteration 0 (baseline)	52
6.4	Iteration 1	54
6.4.1	Iteration 1: Parameters	54
6.4.2	Iteration 1: Evaluation	54
6.5	Iteration 2	56
6.5.1	Iteration 2: Parameters	56
6.5.2	Iteration 2: Evaluation	56
6.6	Iteration 3	57
6.6.1	Iteration 3: Parameters	57
6.6.2	Iteration 3: Evaluation	57
6.7	Iteration 4	58
6.7.1	Iteration 4: Parameters	58
6.7.2	Iteration 4: Evaluation	58
6.8	Refined Monitoring Technique	59
7	Discussion	61
7.1	Containers vs Firecracker	61
7.2	Existing Monitoring Techniques	61

7.3	Improving a Technique for Container Security Monitoring	62
7.4	Evaluation of the Proposed Technique	64
8	Threats to Validity	67
8.1	Construct Validity	67
8.2	Internal Validity	67
8.3	External Validity	68
9	Conclusion	71
9.1	Future Work	72
	Bibliography	73
A	Appendix 1	I

List of Figures

2.1	Architectural differences between Type-I and Type-II hypervisors. . .	6
2.2	Architectural structure of Docker containers.	7
3.1	Iterative process of the design science research methodology.	19
5.1	Example of Sysdig output when attached to a container and logging enter events.	36
5.2	Visualization of the sliding window algorithm using a trace of $t = 11$ system calls, window size $W = 5$ and step size $L = 2$	40
5.3	Architecture of Heimdall.	45
5.4	Architecture of the Hlin framework.	46

List of Tables

4.1	Snowballing procedure information.	21
4.2	Extracted meta-data of included studies.	22
4.3	Extracted miscellaneous data of included studies.	23
4.4	Extracted data describing techniques of included studies.	24
4.5	Extracted evaluations of included studies.	25
4.6	STRIDE coverage and synthesised summary of included studies. . . .	27
4.7	Citation matrix, “X” denotes the row referencing the column, “-“ signifies a similar or later publication date, making a reference unlikely.	31
4.8	Agreement and disagreement for inclusion and exclusion between the review and the test.	31
5.1	Constructing BoSCs using $s = \{2, 5, 10\}$ from the example input sequence.	38
5.2	Creating BoSC from frequency vectors using the example input se- quence with $\Delta T = 100ms$	40
5.3	Using a sliding window with $W = 5$ and $L = 2$ to construct BoSC from the example input sequence.	41
6.1	Hardware configuration.	49
6.2	Software configuration.	49
6.3	Vulnerable containers used in the simulation environment.	50
6.4	Explanations of evaluation metrics.	53
6.5	Baseline technique evaluation results.	53
6.6	Anomaly detection and feature extraction parameters for the first iteration.	54
6.7	Evaluation results first iteration: sliding window feature extraction. .	55
6.8	Evaluation results first iteration: n-gram and frequency vector feature extraction.	55
6.9	Anomaly detection and feature extraction parameters for the second iteration.	56
6.10	Evaluation results second iteration: n-gram feature extraction. . . .	56
6.11	Anomaly detection and feature extraction parameters for the third iteration.	57
6.12	Evaluation results third iteration: n-grams and sliding window feature extraction.	58

6.13 Evaluation results third iteration: The three best performing (FPR-wise) combinations for each feature extraction algorithm managing to detect 21 out of 22 attacks. 58

6.14 Anomaly detection and feature extraction parameters for the fourth iteration. 59

6.15 Evaluation results for the fourth iteration using isolation forest with sliding window feature extraction. 59

6.16 Parameters of the top performing technique. 59

6.17 Detailed comparison of baseline and refined monitoring technique. . . 60

A.1 Raw results from all evaluations I

1

Introduction

The use of virtualisation for running software is becoming increasingly popular. The industry has started a transition towards cloud-native applications, which has caused an explosive growth of software running in virtualised environments [12]. There are two distinct approaches towards achieving virtualisation: Virtual machines and operating system-level virtualisation, also known as containers.

A *Virtual Machine* (VM) emulates an entire operating system (OS) running on top of, or alongside, another OS. These two systems are commonly referred to as the *guest* OS and the *host* OS, respectively. VMs tend to be quite resource-intensive as emulation of software is inefficient, and OS's are complex [35, 72]. The introduction of hardware support [28] and hypervisors [71] have brought optimised performance and reduced overhead. However, the performance impact is still not negligible and needs to be taken into consideration when evaluating virtualisation approaches.

Containers are an alternative approach that provides virtualisation with less overhead compared to VMs. Instead of emulating an entire OS, containers run directly on the host and use functionality in the host kernel to effectively isolate themselves, achieving virtualisation. Thus, containers require support from the kernel to function. Containers are more lightweight than VMs as no emulation is needed, and the containers themselves only consist of an application and its dependencies.

Containers relying on the host kernel means the kernel is shared between the host and all containers running on the host, which introduces a new attack surface not present in VMs. Vulnerabilities in the host kernel can be exploited from within a container to affect entities outside, or break out of, the isolated environment of the container. The new attack surface is a major reason for security being one of the top concerns regarding containers [9].

Monitoring the behaviour of containers to detect potential attacks or otherwise anomalous behaviour is a strategy that can be employed to mitigate parts of the inherent security-related drawbacks introduced by containers. The behaviour of containers manifests itself in multiple ways that can be monitored and analysed, with typical behavioural data sources being *system calls* and *resource utilisation*. System calls are superior to resource utilisation as system calls represent the behaviour in a more fine-grained manner, enabling the detection of anomalous behaviour caused

by sophisticated attacks. Monitoring system calls comes with the added benefit of being a non-intrusive monitoring approach, meaning that it is a drop-in solution and no modifications to the monitored entity are necessary.

Detection of security attacks can be categorised into two subcategories, signature-driven approaches [38, 40] and anomaly-driven approaches [25, 22]. Signature-driven approaches can be likened to the rules of a firewall, where what is allowed or not allowed is specified explicitly. However, for more complex applications, signature-driven approaches tend to become complex and hard to maintain [51]. Furthermore, signature-driven approaches cannot detect unknown attacks, commonly referred to as *zero-day attacks* [51]. On the other hand, anomaly-driven approaches aim to detect behaviour that deviates from what is considered *normal* for the application in question. These approaches can detect many types of attacks, including zero-day attacks, and perform great in domains where the behavioural space is large [51]. However, anomaly-driven approaches require the normal behaviour to be modelled in order to detect anomalies.

When evaluating the performance of a security-focused monitoring technique, it is of high importance to ensure a diverse set of attack scenarios. The STRIDE threat model [33] categorises threat types based on what an attacker is able to achieve. This thesis aims to provide a refined anomaly-driven monitoring technique using system calls as behavioural data, providing an evaluation of its performance using threats from multiple STRIDE categories on a diverse set of commonly containerised applications.

1.1 Problem Domain and Motivation

The use of containers and the technology behind them is considered relatively immature compared to current solutions, mainly VMs. As previously explained, containers do provide certain benefits over VMs, e.g. less overhead and greater flexibility [65]. However, what makes containers lightweight also introduces security-related drawbacks. As containers run directly on the host, the host's kernel is shared between the host itself and all containerised applications. Thus, containers introduce an additional attack vector, the shared kernel, meaning that attackers can potentially exploit kernel vulnerabilities to affect the outside of the virtualised environment or break out of it entirely. Threats such as the one described have led to security becoming one of the top concerns for the use of containers in production environments [9].

Studies have been done to investigate the benefits of container-based environments and deployment [66]. Other studies have investigated the security implications of containerised environments [10, 16]. Identifying and improving current monitoring techniques for container-based systems is a first step towards a safe migration to containers. This study aims to provide further knowledge of how to monitor applications running in containers.

1.2 Research Questions and Research Goals

First, this thesis aims to analyse the current cutting edge container monitoring techniques available. Second, based on the performed analysis, this study explores an improved container monitoring technique that provides more comprehensive security in relation to the STRIDE security threat model [33] and improvements to effectiveness, e.g., detection rate. The improved monitoring technique will be created by using an existing technique as a foundation or by combining multiple techniques to provide additional layers of security.

In this thesis, the following research questions are addressed:

- RQ1:** *What current monitoring techniques exist that might be applicable for the container-based virtualisation environment?* Identify existing techniques which have the potential to be either refined or combined in order to achieve the goals.
- RQ2:** *What are the steps needed to be taken in order to develop an improved monitoring technique with additional layers of security and increased effectiveness?* To gain a fundamental understanding of how container monitoring techniques work and use that knowledge to design and implement a potentially improved monitoring technique.
- RQ3:** *How well does the refined monitoring technique perform with respect to STRIDE model coverage and performance, e.g. ability to detect threats?* The goal is to evaluate and determine if the created technique covers additional attack types and scenarios and investigate how well it performs.

1.3 Contributions

The results of this research are targeted towards both organisations and research audiences interested in making use of the many benefits presented by the container technology. As evidenced by the literature review, the amount of security-focused container monitoring research is lacking. Combining this fact with the ongoing popularity surge of containers makes all research related to the topic a welcome addition. This study contributes to the research community by detailing the creation and evaluation of a novel monitoring technique using the Isolation Forest to detect anomalies in system call logs. The technique was designed based on knowledge attained and evaluations analysed during a literature review. Part of the review process involved data extraction, resulting in multiple tables listing useful information of the examined techniques for use with containers at the time of writing. In addition to the previously mentioned contributions, two tools were developed to facilitate the development and evaluation process. The system call monitoring tool, Heimdall, provides effortless logging and labelling of system calls invoked from containers. The anomaly detector evaluation framework, Hlin, carries the responsibility of performing feature extraction on logs produced by Heimdall, transforming them into suitable data representations. Hlin is also responsible for evaluating the developed monitoring technique. The evaluation aims to inform interested parties of the performance of the developed monitoring technique, presenting the number

of correctly detected attacks and False Positive rates (FPR). The evaluation gains additional value by examining the detection performance using 22 disclosed real-world exploits and vulnerabilities found in software listed as part of the Common Vulnerabilities and Exposures (CVE) database. The refined non-intrusive monitoring technique successfully detects all 22 attacks while keeping the FPR below 3%, a definite improvement over the baseline set as a reference point for evaluation.

1.4 Scope

The scope of this thesis is in the domain of monitoring containers and commonly containerised applications. This study prioritises non-intrusive monitoring techniques to provide a universal container monitoring technique applicable to any container. A non-intrusive approach allows monitoring of any application running in a container, thus expanding the application domain where the refined anomaly detection technique is applicable, increasing its generalisability. A literature review is conducted using the snowballing search approach exploring the currently available container monitoring techniques without requiring the time commitment of conducting a complete systematic literature review. The thesis prioritises evaluating the refined anomaly detection technique using a diverse set of applications vulnerable to real-world exploits at the expense of using synthetically generated over real-world normal behaviour. The refined monitoring technique should, theoretically, also function as an online anomaly detection technique. However, in this thesis, only offline anomaly detection has been implemented and evaluated.

1.5 Structure of the Thesis

The remainder of the thesis is structured as follows; Chapter two contains the background information pertinent for attaining a base-level understanding of the context. Chapter three describes the approach of how this thesis plans to answer the stated research questions found in section 1.2. The literature review results are found in chapter four, including tables summarising the different studies examined, a citation matrix and an analysis of the results. Chapter five details the implementation of the refined monitoring technique, including the artefacts produced during the iterative design science process and the tools developed to facilitate the evaluation of said artefacts. Chapter six contains a description of the created simulation environment, evaluation metrics and the evaluation results. The two subsequent chapters discuss the results and evaluation, followed by the threats to validity and related mitigation strategies. The last chapter presents the conclusion and future work.

2

Background

This chapter will provide an introduction to the most critical topics that are addressed in this report. Furthermore, this chapter will highlight some of the reasons why virtualisation and especially containers are a point of high interest in the field. The first and second sections cover virtualisation in general and how containers can prove beneficial over other means of virtualisation. The following section provides information about security and the threat landscape surrounding container technology. The following section provides a background to container security protection and outlines a few different means of achieving security and their differences. The chapter concludes with related work on the topics previously discussed.

2.1 Virtualisation

The core concept of virtualisation is to create abstraction layers of computer hardware on which computer systems can run, making the systems *virtual*. Development of the technique began in the 1960s with the primary goal of a logical division of system resources to run multiple applications simultaneously on a single machine [49]. The technique has since seen further development and can provide additional benefits such as isolation, snapshots, and hardware independence.

The popularity of virtualisation took off in the early 2000s as enterprises realised the benefits of partitioning their servers to allow a single server to run multiple legacy applications with different dependencies and optimise server utilisation to reduce expenses. Furthermore, the last decade has seen the industry starting a transition towards cloud-native applications and cloud computing, which has caused an explosive growth of software running within virtualised environments in data centres [12].

Traditionally, two conceptually different approaches have existed to achieve virtualisation: Virtual machines and operating system-level virtualisation, also known as containerisation. Although similar, the approaches offer environments with different characteristics, benefits, and drawbacks. There have been recent developments to create a new approach without the drawbacks of the traditional approaches, e.g., Firecracker developed at Amazon Web Services introduced in section 2.1.3.

2.1.1 Virtual Machines

A virtual machine (VM) is an emulation of an entire operating system running on top of, or alongside, the main OS, commonly referred to as a *guest* OS and *host* OS, respectively. While VMs provide good isolation, as evidenced by their extensive utilisation in cloud computing services such as Amazon Web Services [3, 8], they tend to be relatively inefficient as software emulation introduces overhead and OS's are complex [35, 72]. However, the issue has been partly resolved with the introduction of hardware support [28, 6], hypervisors [71] and more recently, tools like Firecracker [3]. A hypervisor is a software, firmware, or hardware layer that manages the creation and execution of VMs. Hypervisors can be classified into two types, Type-I and Type-II, depending on their underlying architecture (see figure 2.1). Type-I, also called bare-metal, hypervisors run directly on the host's hardware. Type-II, also called hosted, runs on top of an OS just like any other conventional application.

2.1.2 Containerisation

Containerisation, or operating system-level virtualisation, is a virtualisation approach where the operating system kernel provides tools that enables the dynamic creation of isolated user spaces. In Linux, and from now on in this thesis, these spaces are referred to as *namespaces* [42].

Namespaces "*wraps a global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource.*" [42]. Hence, they enable the Linux kernel to handle the logical isolation of processes, achieving virtualisation. Linux has multiple namespace types (e.g. network, PID, mount), providing flexibility in the configuration of virtualised environments. i.e. we can isolate processes using the PID namespace while still allowing them access to the same file system by omitting the mount namespace. The

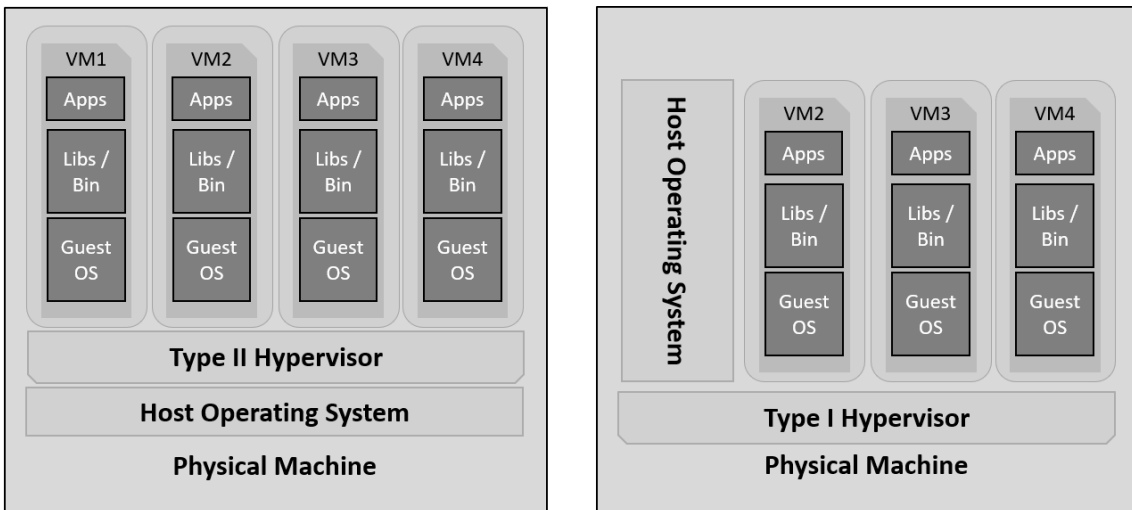


Figure 2.1: Architectural differences between Type-I and Type-II hypervisors.

and an API to configure and manage MicroVMs. The virtualisation technology can run MicroVMs with negligible CPU overhead and memory overhead as low as 3%, and provides excellent isolation with protection against sophisticated side-channel attacks (e.g. Spectre [32]) and fast boot times (150ms).

2.2 Docker

Docker, also called Docker Engine, is one of the most prominent container technology platforms designed to ease the development, deployment and execution of applications using containers. The platform uses a client-server architecture consisting of three components, the Docker daemon, Docker client, and Docker registries. The Docker daemon runs on the host system and handles everything related to the container life cycle, i.e. building and running containers and setting up namespaces. The Docker client's purpose is to interact with and issue commands to the Docker daemon and is usually a command-line interface (CLI). The Docker registries store Docker images for the purpose of serving them to Docker daemons if needed. Docker Hub is the official public Docker registry. However, other both private or public registries can be used [18].

Docker creates an abstraction of the relatively low-level nature of namespaces and provides means of configuring, building, and bundling applications and their dependencies. Using the power of Linux namespaces and cgroups, it is capable of isolating applications with great flexibility.

Additional tools have been created for Docker to decrease the complexity of setting up containerised applications. One such tool is **docker-compose**, which is used to define and run multi-container Docker applications. It provides a syntax to configure multiple interconnected containers using a single configuration file and has been extensively used throughout this thesis to streamline the development.

2.3 Threat Landscape and Security

The rise of container technology introduced a new threat landscape, as the host-container domain differs from the domain of previous generation's applications, e.g. VMs. Sultan et al. [68] derived four generalised use-cases which cover every aspect of the host-container threat landscape. The use cases include; (I) *protecting a container from applications inside it*; (II) *inter-container protection*; (III) *protecting the host from containers*; and (IV) *protecting containers from a malicious or semi-honest host*. They found that the first three use cases can be remedied by software-based solutions, while the last use case requires a hardware-based solution. This thesis focuses on software-based solutions, meaning only the first three use-cases are relevant.

An attack vector in the new threat landscape is that containers share the host kernel, meaning vulnerabilities in the kernel can be abused from the inside of the *isolated* container. Before the widespread utilisation of containers, when applications ran

inside their own VMs or on separate systems, kernels were not shared and could not be exploited to compromise other, isolated applications.

2.3.1 STRIDE

To help understand the threat landscape and subsequently understand the capabilities of any particular monitoring technique, the STRIDE security threat model, as defined by Kohnfelder and Garg [33], has been employed. The model helps with categorising different types of threats and aids the process of understanding the characteristics a monitoring technique should have to cover a specific type of threat. STRIDE consists of the following types:

Spoofing - When an attacker manages to identify as someone else by falsifying identifying information, gaining an unintended advantage or illegitimate access.

Tampering - When an unauthorised actor can modify data through unintended or illicit means, e.g. editing the contents of a message in flight or removing/editing important information on a system.

Repudiation - When the origin of a message or statement can successfully deny being the origin. Ensuring *Non-repudiation* allows for validation of the signature and origin, guaranteeing such denials can be proven false.

Information disclosure - When an attack achieves privacy breach, data leak or accessing confidential data. Covering this threat type would mean no such unauthorised access should be possible.

Denial of service - Denial of service refers to an attackers ability to, through illicit means, manage to deny its intended users access to a system, service or data.

Elevation of privilege - When an attacker manages to acquire additional permissions by illicit and unintended methods [33].

This model was designed with Microsoft's products in mind. While the classifications are not product specific, containers and software running in containers differ from traditional software.

2.4 Threat Protection Approaches

Threat protection can be achieved using different approaches; some reduce the attack surface, others detect anomalous behaviour or vulnerable code. However, no one approach is complete, and they all serve different purposes towards the same unified goal of security. This section describes some existing approaches which can be employed to defend against or mitigate threats to containerised systems and applications.

2.4.1 Static Analysis

Static analysis of software can be performed on either source code or an already compiled executable. In the context of containers, the target of scanning would be the container image. Regardless of the type of target, the scanner compares the target with known vulnerabilities and can, as a result, only detect known vulnerabilities. Clair is a static image scanning tool for containers that match packages and their versions with remote Common Vulnerabilities and Exposures (CVE) databases. However, as research by Tunde-Onadele et al. [70] shows, Clair did not detect vulnerabilities in more than three out of the 28 containers they used in their research.

2.4.2 Behavioural Analysis

A compromised application often causes the behaviour of the application to deviate from its normal. Monitoring the behaviour and analysing it for anomalous activity is an effective way of detecting an ongoing attack. The behaviour of applications often manifests itself through either resource utilisation or system calls, both of which can be analysed for anomalous activity using either statistical or rule-based approaches.

2.4.2.1 Machine Learning-based Analysis

Machine learning (ML)-based analysis of data is an automated approach to statistical analysis where an algorithm is fed data to create an *internal representation* of what the algorithm learned about the data, also called a model. The produced model can be used to classify or perform predictions on new, previously unseen data. ML is a powerful tool as the algorithms used can *learn* the correlations and patterns in data too complex or vast for a human to analyse, making it a great tool to learn an application's normal behaviour and detect anomalies.

Machine learning algorithms are commonly split into two categories, supervised and unsupervised learning. Supervised learning is generally used for classification and unsupervised learning for outlier detection and clustering [36]. Laskov et al. [36] compare different learning algorithms for both categories and their resulting ability to classify known as well as unknown threats correctly. They conclude that the supervised learning methods outperform the unsupervised ones when only dealing with known attacks. However, the performance gap diminishes when unknown attacks are introduced, leaving the unsupervised methods more practical since they do not require labelled data.

Measuring the performance of a classification algorithm on a specific set of data is commonly performed using *confusion matrices*. A confusion matrix is a table composed of the algorithm's predictions and the true values of the data, which can be used to derive a plethora of performance-indicating metrics, e.g. True Positive rate or F1 score.

2.4.3 Runtime Security

Enhancing the runtime security of containerised applications is performed by limiting their capabilities after startup, per the *principle of least privilege*. Restricting an application’s access to only the files, system calls, and other system resources needed for its legitimate purpose is an effective way of mitigating the potential impact of a successful attack. Reliable tools such as AppArmor, seccomp and Sysdig Falco exist for this purpose and have proven themselves to have potential [14].

2.5 Related Work

The idea of monitoring system calls to detect anomalous behaviour dates back to before the 2000s. Forrest et al. proposed using the system calls and their sequence to distinguish anomalous behaviour [22]. System call anomaly-detection methods can be divided into two subgroups; frequency-based and sequence-based. The former completely ignores incoming system calls and instead records and stores the frequency of each distinct system call. The sequence-based approach instead keeps track of the sequence of the system calls, storing said sequences for later use, notably requiring more storage compared to the frequency-based counterpart.

Bag of System Calls (BoSC) is a frequency-based system call representation of system behaviour introduced by Kang et al. in 2005 [29] and could be viewed as an evolution of what Forrest et al. proposed almost a decade earlier [22]. Kang et al. define the bag of system calls as an ordered list $\langle c_1, c_2, \dots, c_n \rangle$ where the total number of distinct system calls is denoted as n . c_i is the number of occurrences of said system call. These bags can then be used in combination with machine learning techniques to detect anomalies of system behaviour.

Another technique making use of the system calls, however focusing the sequence rather than frequency, is the Sequence Time-Delay Embedding (STIDE) technique introduced by Forrest et al. [22] published in 1996. STIDE uses a database of short sequences, size k , to define what is to be considered normal. The database is populated by sliding a window of size $k + 1$ over a healthy systems trace, storing the system call sequences. STIDE was proposed almost 30 years ago and have since seen some improvements in 1998 [27] and in 1999 by Warrender et al. [77]. Abed et al. call STIDE “a simple and efficient technique” [1] yet call attention to the growing rate of the database potentially being linear with the number of system calls in the trace.

A technique similar to STIDE [22] is the Sliding Window technique proposed by Lee and Stolfo in 1998 [37]. As the name suggests, their proposed technique also makes use of a sliding window. However, the size of the windows and sliding step differs. Lee and Stolfo propose a window size of $2l + 1$. The window slides over the sequence of stored system calls with a sliding step of l . The stored sequences are then analysed using RIPPER [15] labelling regions of the sequences as either abnormal or normal. A signal is sent if the percentage of abnormal vs normal regions exceeds a certain threshold.

Hidden Markov Models (HMM), together with the sequence of system calls, has been used extensively to create intrusion detection systems [75, 77, 26, 13, 81]. While these all use sequence-based system calls to train their HMM classifier, their implementation differs. Warrender et al. [77], and Wang et al. [75] both make use of probabilities, where Warrender signals anomaly if the probability of one system call within a sequence is below a defined threshold. Wang instead looks at the entire sequence and signals when the probability of said sequence falls below a certain threshold. Hoang et al. [26] present a multi-layered approach combining the HMM with the sliding window. The final outcome is the combination of the two techniques outcomes. Cho and Park's implementation [13] is more focused. Their proposed system only considers system calls generated by root privilege operations, announcing a 100% Detection Rate (DR) and an FPR between 3-22%.

Warrender et al. compared HMM, STIDE and RIPPER-based methods in [77]. While all techniques performed adequately, they concluded HMM provided the best detection rate and low minimum false positives at the cost of high computational demand. Areeg and Claus apply HMM in [7] to detect and identify anomalies in containerised cluster environments, monitoring resource utilisation and response times instead of system calls, reporting 96% precision. Additionally, they use Hierarchical Hidden Markov Models (HHMM) to localise the anomalies, successfully doing so in 97% of evaluated cases.

Alfari and Wolthusen implemented a host-based intrusion detection system for virtual machines part of a multi-tenancy Infrastructure-as-a-service (IaaS) environment using BoSCs [4]. Their implementation monitors system calls between the VM and the host operating system, treating the VM as a single process. They combined BoSC with the sliding window technique. However, the sliding window technique is not used to track the sequence; instead, it is used to receive new system calls and drop old ones. This is done to lower the amount of tracked system calls when possible. Their implementation divides the input trace into epochs, applying a sliding window to each epoch to traverse the systems calls of the said epoch. First to add the bags of system calls of a normally behaving system to a database. This database holds frequencies of bags of system calls to act as a reference of *normal* behaviour, i.e. training the classifier. Once trained, the same method was used to gather BoSC and compare them to the previously generated database, declaring an epoch anomalous if the change of BoSC frequencies exceeds a set threshold. Their evaluation reports 100% accuracy, with 100% detection rate with 0% false positive rate (FPR), using a sliding window of size 10.

Abed et al. [2] propose a technique similar to Alfari and Wolthusen's [4] to docker containers instead of VMs. By employing a background service running on the host kernel to monitor the system calls between any docker containers and the host kernel. This is achieved using `strace` to trace all system calls made by the container. This tool collects the origin process ID, arguments and return values. Collecting system calls using `strace` requires no changes to the containers themselves. In [2] all detection and evaluation were done offline. They are reporting accuracy of 100% and FPR at 0.58% using a window size of 10. Abed et al. have since applied the

described technique to monitor systems in real-time [1], reporting a True Positive Rate (TPR) of 100% and a False Positive Rate (FPR) of 2%.

Srinivasan et al. [67] argue that only tracking the frequency of system calls results in lost information necessary to detect some anomalous behaviour. Their technique makes use of n-grams to store and compare the behaviour derived from system calls. Signalling decisions are based on probabilities. They evaluate the performance of Maximum Likelihood Estimator (MLE) and Simple Good Turing (SGT). They conclude the MLE performance superior when only a small number of traces are available for training, SGT otherwise.

Zou et al. [82] monitor the levels of system resource utilisation to detect anomalies in containers by employing a technique based on isolation forests, modified to improve performance in a containerised environment. Data collection is done by a monitoring agent on each host that collects system resource utilisation data. The collected data is sent and stored continuously, only keeping data from the most recent period. The anomaly detection module uses the received data with an iForest-based abnormality evaluation method optimised for containers. Since the resource utilisation fingerprint can vary greatly depending on the type of workload the container is assigned, Zou et al. propose a self-learning bias algorithm to better suit containers' varying nature and workload by weighting the impact of different resources depending on the fingerprint. Performance was improved further by filtering the logs of information not pertinent for anomaly detection and increasing the monitoring frequency when an anomaly is detected. They achieved a detection rate of 72-100% and a false alarm rate of 2-12.2% depending on both types of anomaly and applications running within the containers. When an anomaly is detected, the corresponding log is sent to another module for analysis with the goal of locating the cause of the anomaly.

Ravichandiran et al. [54] monitor resource utilisation to detect anomalies with the intent of mitigating eDoS attacks. They use historical microservice datasets of "normal" behaviour to build autoregressive statistical models. These models were then used to forecast expected resource behaviour, signalling an anomaly of the difference between the generated forecast and the actual behaviour surpass a certain threshold.

3

Approach

In this section, the approach adopted to address the research questions is described. First, a literature review is conducted using the snowballing search approach to explore the available state of the art monitoring techniques and build a foundation of knowledge used to create a refined monitoring technique. After that, the design science research methodology is followed to refine the container monitoring technique. Finally, a simulation environment is designed and implemented to evaluate the refined technique.

3.1 Snowballing Search Approach

With the intent of keeping a reasonable scope of the review, the snowballing search approach based on Wohlin’s guidelines [79] was used. During this process, a better understanding of the problem space was attained in addition to the main objective of answering the first research question by identifying suitable techniques to improve further.

The remaining parts of this section define the review protocol and the methods used to conduct the snowballing search approach. It is necessary to define the protocol before conducting the review to reduce any potential biases, e.g., the selection of individual studies being influenced by the researchers’ expectations.

3.1.1 Tools and Traceability

Two tools were used to assist the described workflow and procedure; Mendeley Reference Manager [48] and Google Sheets. These tools provided a shared database of references, their inclusion/exclusion status and allowed for creating a structure to assist traceability in terms of origin and categorisation for each study.

The primary functionality provided by Mendeley, and thus its purpose, was to have a single location for storing and accessing studies. Mendeley does provide the ability to group studies hierarchically. However, something more sophisticated was deemed necessary to provide traceability for the review process. Thus, a custom spreadsheet was set up, which enabled full traceability of all studies reviewed. The spreadsheet was extended with some functionality to speed up the process, e.g. automatic detec-

tion of already reviewed studies and statistics.

3.1.2 Search Strategy

The snowballing approach [79] was used in order to conduct the search. The approach consists of an iterative process that uses a set of studies to identify additional studies to include in the review.

The first step of the snowballing process is to identify a tentative start set. The tentative start set was generated by a database search, which requires a search string to be crafted. The database chosen was Google Scholar, a choice made to minimise potential bias favouring any single publisher. The search string was derived from the research questions and consists of relevant keywords and combinations. Multiple iterations of search strings were crafted and piloted in order to ensure wide coverage and relevant results. The final search string was the following: *(monitor* OR analysis) AND security AND (attack* OR threat*) AND (anomaly OR misuse OR intrusion) AND (application* OR service*) AND (software OR linux) AND (container* OR docker) AND evaluat**. It returned ~14 000 total results, which was deemed reasonable.

Each result from the database search was either excluded or tentatively included solely based on its title and abstract and the selection criteria defined in section 3.1.3. Results that could not be excluded based on the criteria were added to the tentative start set until the set reached a satisfactory size.

The tentative start set was then condensed into the actual start set by applying the selection criteria to the full studies. The authors performed this process individually, each using different halves of the studies, and then cross-checked the other author's process to ensure the understanding of the criteria were aligned. In order to get a starting set of reasonable size and good quality, some additional thought is required. Wohlin states that a start set of good quality has the following characteristics: [79]:

- Includes studies from different communities.
- Included studies are diverse in regards to authors, years and publishers.
- Prioritises highly cited studies.

Thus, studies in the start set were identified with the goal in mind to achieve these characteristics. The size of the starting set is arbitrary, but Wohlin suggests that if the area being studied is broad, a larger size should be used (and vice-versa) [79].

Once the start set had been identified, the iterative process of the snowballing approach began. Every iteration consists of backward and forward snowballing on a set of studies. For the first iteration, the previously identified start set is used. For subsequent iterations, the set of newly identified studies from the previous iteration is used. Backward snowballing is the process of looking at the references list of the study being examined. In contrast, forward snowballing is performed by identifying all studies which cites the study being examined. All identified studies

are initially excluded if they do not meet the necessary criteria for inclusion or tentatively included, based on the information available in the study being examined. Once a study was tentatively included, the full study was attained and analysed in order to, with certainty, determine if the study should be included or excluded. The analysis consists of first reading the abstract and then the whole study until a decision can be made.

Once an iteration was completed, the set of newly identified studies was used for the next iteration. If a study was found that had already been categorised, it was ignored. The snowballing process was re-iterated until no more studies were found.

3.1.3 Study Selection Criteria

The study selection criteria to be used:

- Include studies proposing techniques for monitoring the security of containers within any application domain.
- Exclude studies not written in English.
- Exclude studies lacking an evaluation of the technique.

When a study, or variations of it, has been published multiple times, the most comprehensive version will be used.

3.1.4 Study Selection Procedure

The researchers performed the initial evaluation of studies individually to mark them as excluded or tentatively included. The tentatively included study was then evaluated and discussed by both researchers before finally including or excluding said study. The selection procedure was performed like this to speed up the snowballing process while still ensuring unanimous application of the selection criteria. The risk of incorrectly excluding a study during the initial (individual) evaluation is considered low, as any doubt about the study would result in it being tentatively included.

In order to verify the quality of the review, an inter-rater reliability test was performed using Cohen's kappa coefficient [47]. The test was conducted by asking a third party to randomly select 10% of the studies included in the starting set and let them perform the snowballing procedure according to the method specified in this chapter. Due to the sheer amount of work involved, the test was limited to a single iteration. The resulting studies and their inclusion/exclusion categorisation was then used to calculate Cohen's kappa coefficient.

3.1.5 Data Extraction and Synthesis

In order to get an overview of the identified techniques, which would guide the selection of techniques to be used as a foundation for the refined technique created in this thesis, a data extraction process was performed. In order to extract relevant

data systematically, with minimal risk of introducing bias, a data extraction form was constructed and piloted before conducting the review. The final form includes the following information:

- Title
- Source (e.g. the conference or journal)
- Author(s)
- Year of publication
- Hardware used for evaluation
- Software used for evaluation
- Container technology
- Source of data used for analysis (e.g. system calls, resource utilisation)
- Analysis method/algorithm(s) (e.g. n-grams, Nearest Neighbour)
- Evaluation results
- STRIDE security coverage
- Contextualised qualitative summary

During the piloting, some observations were made; (I) Some parts of the form are qualitative and require interpretation; (II) Some studies will not include all information due to low quality or the approach taken; (III) There are many different ways to perform and present evaluations. These observations impose the need for qualitative analysis that can be contextualised into a format more appropriately suited for comparison, which led to the inclusion of a qualitative summary to the form. To mitigate the potential source of bias introduced by a qualitative summary, all summaries were written by one author and confirmed by the other.

The extracted data were tabulated and ordered alphabetically by the first author name to provide an overview of all techniques found. The tabulated data were then reviewed to identify which techniques are suitable to be used as a foundation for the thesis. The data was extracted by both researchers together.

3.2 Refining a Monitoring Technique

To refine a monitoring technique, the *design science research methodology* (DSRM) will be employed. DSRM is not used to prove a theory, but to *refine* it by exploring why, when and how a solution works [20]. It is an iterative process that can be divided into three major steps, also illustrated in figure 3.1:

1. Understanding/exploring the problem space
2. Ideation and design of solution (artefact)
3. Implementation and evaluation of the solution

The methodology used in this thesis is primarily based on the *Regulative cycle* described by Wieringa [78] as part of the Design Science Research Methodology (DSRM).

Understanding/exploring the problem space.

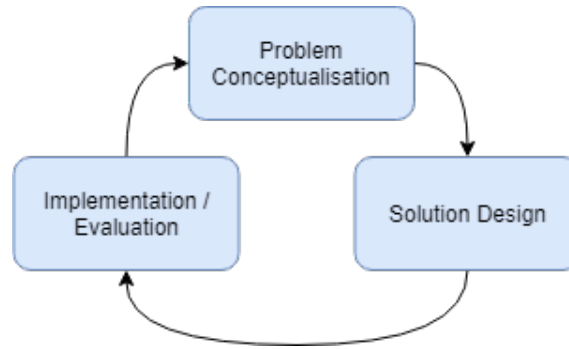


Figure 3.1: Iterative process of the design science research methodology.

The first step calls for a deeper understanding of the problem intended to be solved. Wieringa [78] refers to this process as *problem investigation*, the emphasis of which depends on the reason behind the investigation. Using the categories listed by Wieringa, this thesis falls under *Goal-driven investigation*. The problem must be understood to a satisfactory degree before it is wise to begin the design process. The review detailed in 3.1 intends to serve as the problem investigation of this thesis. Guidelines of DSRM often mention the importance of communication with the stakeholders throughout the entire process [31, 78]. However, given the nature of this project and the absence of involved companies, the authors and the supervisor acts as stakeholders.

Ideation and design of solution (artefact).

This step involves designing a solution based on the problem investigation. Said final design will act as an answer to RQ2. This process is referred to by Wieringa as the *solution design*. The term solution is optimistic as there is no guarantee that the designed artefact actually solves the identified problem but rather intends to do so.

Before implementation, a validation of the design is performed. Usually, the step involves stakeholders by asking whether the proposed solution design would bring them closer to their goal if implemented correctly. Wieringa specifies three questions that should be asked in order to validate the design [78]:

Internal validity. Does the proposed design satisfy the criteria identified in the problem investigation?

Trade-offs. How would slightly modified designs satisfy the criteria?

External validity. Would the design satisfy the criteria, if implemented in a slightly different context?

As this research does not involve any external stakeholders, the purpose of the validation would be more of a sanity check in order to detect detrimental mistakes in the design. However, a validation will not be conducted, and this thesis will instead rely on the literature review to provide a foundation that has already been validated through previous research.

Implementation and evaluation of solution.

The final steps involve creating a simulation environment and the implementation and evaluation of the designed artefact. The simulation environment is designed to generate data that are necessary for the evaluation. The source data is the trace of system calls logged during the execution of normal and anomalous behaviour of containers running software versions with known vulnerabilities. The Common Vulnerabilities and Exposures (CVE) databases provide extensive lists of identified vulnerabilities to use for this purpose. Evaluating the detection capabilities of real-world vulnerabilities adds significance to the results. In order to properly evaluate the improvements, a baseline using the same data must first be generated, which is achieved by implementing an existing technique including source data, feature extraction and the detection algorithm. The metrics of interest are False Positive Rate (FPR) and if the attack is detected or not. The next step is to implement the refined technique, possibly by using existing techniques as a foundation and evaluating the technique using the same data set and metrics.

Due to the iterative nature of the cyclic process described, this evaluation acts as a foundation for the problem investigation of the next cycle.

4

Results of Literature Review

In this chapter, the results of the conducted literature review using the snowballing approach are presented. Due to the sheer amount of data, it has been split into a series of tables representing different aspects of the data. The chapter concludes with an analysis of the extracted data and the resulting conclusions made regarding designing a refined monitoring technique.

4.1 Snowballing

The information regarding the snowballing procedure is presented in table 4.1. In total, four snowballing iterations were performed. The initial database search conducted to establish a start set returned $\sim 14\,000$ total results, which meant that some stopping criteria had to be used given the infeasibility of examining every result. The stopping criteria used was *when a page of returned results yields no tentatively included studies*, where one page consisted of ten results. In total, ninety studies were examined during the creation of the start set, meaning that the ninth page of results exclusively yielded excluded studies.

Iteration	Examined studies	Included studies	Efficiency
Start set	90	9	10%
Iteration 1	465	8	1.7%
Iteration 2	342	2	0.6%
Iteration 3	60	1	1.7%
Iteration 4	15	0	0%
Total	972	20	2.1%

Table 4.1: Snowballing procedure information.

4.1.1 Extracted Data

The extracted and synthesised data from the included studies have been split into five separate tables. Table 4.2 presents the meta-data of the included studies, table 4.3 miscellaneous information, table 4.4 information about the proposed techniques, table 4.5 the evaluation results and table 4.6 the synthesised data, i.e. STRIDE

4. Results of Literature Review

coverage and the contextualised summary. The included studies have been assigned an identifier which is consistent across all tables.

Table 4.2: Extracted meta-data of included studies.

ID	Ref	Author(s)	Year	Source
T1	[2]	Abed et al.	2015	2015 IEEE Globecom Workshops (GC Wk-shps)
T2	[1]	Abed et al.	2016	International Workshop on Security and Trust Management
T3	[5]	Aljebreen	2018	Florida Institute of Technology
T4	[17]	Cui	2020	Auburn University
T5	[19]	Du et al.	2018	International Conference on Algorithms and Architectures for Parallel Processing
T6	[23]	Fourati et al.	2019	2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)
T7	[24]	Gantikow et al.	2020	Communications in Computer and Information Science
T8	[30]	Karn et al.	2021	IEEE Transactions on Parallel and Distributed Systems
T9	[39]	Li et al.	2019	ICCDA 2019: Proceedings of the 2019 3rd International Conference on Compute and Data Analysis
T10	[41]	Lin et al.	2020	ACSAC '20: Annual Computer Security Applications Conference
T11	[46]	Lu et al.	2019	Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS
T12	[54]	Ravichandiran et al.	2018	2018 4th IEEE Conference on Network Softwarization and Workshops, NetSoft 2018
T13	[7]	Samir and Paul	2020	Free University of Bozen-Bolzano
T14	[67]	Srinivasan et al.	2018	Proceedings - 2016 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI 2016
T15	[69]	Sun et al.	2020	Cloud Computing – CLOUD 2020
T16	[70]	Tunde-Onadele et al.	2019	2019 IEEE International Conference on Cloud Engineering
T17	[74]	Wang et al.	2018	Future Generation Computer Systems
T18	[76]	Wang et al.	2020	IEEE Transactions on Industrial Informatics
T19	[80]	Ye et al.	2018	Cloud Computing – CLOUD 2018
T20	[82]	Zou et al.	2019	IEEE Transactions on Cloud Computing

Table 4.3: Extracted miscellaneous data of included studies.

ID	Hardware	Software	Container technology
T1	N/A	Ubuntu Server 14.04 MySQL 5.6	Docker
T2	N/A	Ubuntu Server 14.04 MySQL 5.6	Docker
T3	N/A	N/A	Docker
T4	Intel Xeon E5-2650 64GB memory	Ubuntu 16.04	Docker
T5	4 vCPU 8GB memory	cAdvisor Heapster InfluxDB Grafana	Kubernetes
T6	4 vCPU 8GB memory	Sysdig Elasticsearch Kafka	Kubernetes
T7	N/A	Debian GNU/Linux 9.5 (stretch) Docker 18.06.1-ce Sysdig 0.24.1 Falco 0.13.0	Docker
T8	4 vCPU 16GB memory	Ubuntu 16.04	Kubernetes
T9	N/A	N/A	N/A
T10	2 vCPU 8GB memory	Ubuntu 16.04	Docker
T11	Host: 8 vCPU 16GB memory Containers: 1 vCPU 2GB memory	Ubuntu 16.04 Docker 18.03-ce	N/A
T12	2 vCPU 5GB memory 20GB disk	Ubuntu 16.04 Nginx	Docker
T13	3 vCPU 2GB memory	Ubuntu 18.04.3	Docker
T14	N/A	Ubuntu 18.04.3	Docker
T15	Intel Xeon E5-2630	Ubuntu 18.04 Docker 1.40	Docker

4. Results of Literature Review

ID	Hardware	Software	Container technology
T16	2GB memory 40GB disk	Ubuntu 16.04 Docker 17.05.0 Sysdig 0.19.1 Clair 2.0.0	Docker
T17	Intel i5-3470 16GB memory	MySQL Zabbix	Docker
T18	Intel Xeon E5-2609 32GB memory	CentOS 7.2.1511 Docker 18.03.0-ce	Docker
T19	N/A	N/A	Docker
T20	Intel Xeon E-5620 32GB memory	Ubuntu 16.04 Docker 18.03.1-ce Logstash 6.2.4 InfluxDB 0.13.0 MySQL 5.6 Memcached 1.5.7 CloudSuite 3.0	Docker

Table 4.4: Extracted data describing techniques of included studies.

ID	Data source	Analysis method/algorithm(s)
T1	System calls	Frequency-based BoSC with sliding window. Detects BoSC, which does not exist in the normal database (mismatch) and flags anomaly if the number of mismatches exceeds a threshold during an epoch.
T2	System calls	Frequency-based BoSC with sliding window. Detects BoSC, which does not exist in the normal database (mismatch) and flags anomaly if the number of mismatches exceeds a threshold during an epoch.
T3	System calls	Frequency-based BoSC with sliding window. Models tested: IQR, k-NN, ANN, C4.5 Decision Tree, Random Forest, SVM
T4	System calls	Sliding window + LSTM autoencoder
T5	System resource metrics	SVM, Random Forest, Naive Bayes, Nearest Neighbour. DTW algorithm to locate anomalous container.
T6	CPU usage Memory usage	Modified Z-score to detect outlier pods. Decision tree to identify root cause of anomaly.
T7	System calls	Sysdig Falco (rule-based)

ID	Data source	Analysis method/algorithm(s)
T8	System calls CPU usage	Non-overlapping n-grams of system call sequences. Models tested: Decision Tree, XgBoost ensemble, Feed-forward vanilla ANN, Feedback RNN
T9	System resource metrics	Random Forest
T10	System calls	Autoencoder reconstruction error
T11	Component response times	Graph similarity
T12	CPU usage Network usage	Autoregressive models predicting normal behaviour to detect abnormal behaviour.
T13	Response latency Resource utilization	Hierarchical hidden Markov model
T14	System calls	Calculates the probability of a n-gram of system calls occurring. Methods tested: Maximum Likelihood Estimator, Simple Good Turing
T15	Resource utilization	Prophet (forecasting procedure) used to predict normal behaviour to detect abnormal behaviour.
T16	Resource utilization System calls Static analysis of containers	System call frequency vectors and system call time vectors. Models tested: k-NN, PCA + k-NN, K-means, Self-organizing map, Clair (static analysis tool)
T17	Resource utilization	Principal Component Analysis (PCA) Eigenvectors
T18	System calls perf (Linux performance data)	EnsembleVAE
T19	Feedback data Status information Performance data	Quantile regression
T20	Resource utilization	Optimized Isolation Forest

Table 4.5: Extracted evaluations of included studies.

ID	Evaluation
T1	TPR: 100%, FPR: 0.58% Higher threshold reduces FPR to 0%
T2	TPR: 100%, FPR: 2%

4. Results of Literature Review

ID	Evaluation
T3	No dimensional reduction (Dimensional reduction applied): IQR: Acc: 92%, TPR: 90%, FPR: 6% k-NN: Acc: 87%, TPR: 82%, FPR: 10% ANNs: Acc: 97% (97%), TPR: 97% (97%), FPR: 1.3% (1.3%), Process time: 13.28s (13.28) C4.5 Decision Tree: Acc: 99% (98%), TPR: 99% (98%), FPR: 1.4% (2.8%), Process time: 1.87s (0.14s) Random Forests: Acc: 100% (99%), TPR: 100% (100%), FPR: 0% (2%), Process time: 3.52s (0.62s) SVM: Acc: 100% (98%), TPR: 100% (98%), FPR: 0% (2.4%), Process time: 1.45s (0.47s)
T4	Multiple combinations of models and parameters are tested against multiple attacks. Accuracy between 76% and 95% depending on the type of attack
T5	Tested on three datasets. F1 Score ranges: SVM: [0.77, 0.93] RF: [0.91, 0.99] NB: [0.78, 0.97] kNN: [0.93, 0.97]
T6	Better response times than standard Kubernetes deployment and manages to identify the attack
T7	Specifies a list of scenarios that they can detect using Sysdig and Falco. Only evaluates the overhead created when using or not using a filter
T8	Decision Tree: Accuracy: 97.1%, 2.7s training, 0.02s prediction time, 97% CPU, 243MB memory XgBoost: Accuracy: 89.4%, 18.7s training, 0.25s prediction time, 165% CPU, 367MB memory FFw Vanilla ANN: Accuracy: 79.7%, 35.1s training, 2.0s prediction time, 335% CPU, 182MB memory RNN + LSTM Autoencoder: Accuracy: 78.9%, 1340s training, 7.63s prediction time, 385% CPU, 242MB memory
T9	Original Dataset: Accuracy 0.846, Sensitivity: 0.811, Precision: 0.879, MCC: 0.701 Labeled Dataset: Accuracy: 0.861, Sensitivity: 0.836, Precision: 0.884, MCC: 0.729
T10	Application Classification: TPR: 91%, FPR: 0.29% Anomaly Classification: TPR: 74%, FPR: 0.24%
T11	Precision: 0.9, Recall: 0.8
T12	Manages to detect Anomalies based on CPU Util and Packets/s, comparing normal data vs current
T13	Precision: 96% for moderate data set. Root cause accuracy: 97%
T14	MLE: Accuracy: 96-99.4%, Sensitivity: 96-100%, Specificity: 86.6-100% SGT: Accuracy: 89-98.7%, Sensitivity: 78-99.8%, Specificity: 66-92.7%
T15	TPR: 100%, FPR: 0%

ID	Evaluation
T16	Clair: 10.71% TPR k-NN: 32% TPR, 9.9% FPR, 0.57s detection time PCA + k-NN: 36% TPR 9.9% FPR, 1.00s detection time K-means: 68% TPR, 7.7% FPR, 0.36s detection time SOM time: 75% TPR, 1.9% FPR, 25.77s detection time SOM freq: 79% TPR, 1.7% FPR, 28.73s detection time
T17	Reduces detection time (alarm delay) by 39.1%. In the case of memory leak: 68.7%
T18	ROC=(90%, 99%)
T19	CPU, Memory and Disk Attacks: CNN: 60% Precision, 100% Recall, 91.3% Acc, RNN: 60% Precision, 100% Recall, 91.3% Acc, BRNN: 75% Precision, 100% Recall, 95.7% Acc, DRNN: 75% Precision, 100% Recall, 95.7% Acc DDoS Attack: Incoming Traffic: 100% Precision, 100% Recall, 100% Acc, Outgoing Traffic: 90% Precision, 100% Recall, 98.94% Acc
T20	Memcached: Endless loop CPU: 100% DetectRate, 2% FPR Memory Leak: 98% DR, 2% FPR Disk I/O Fault: 76% DR, 5% FPR Network Congestion: 100% DR, 2% FPR Web Search: Endless loop CPU: 100% DetectRate, 5.7% FPR Memory Leak: 100% DR, 7.4% FPR Disk I/O Fault: 72% DR, 12.2% FPR Network Congestion: 84% DR, 6.7% FPR

Table 4.6: STRIDE coverage and synthesised summary of included studies.

ID	STRIDE	Summary
T1	TIDE	Offline testing for detection of anomalies (not real-time). No performance evaluation.

4. Results of Literature Review

ID	S T R I D E	Summary
T2	T I D E	Targets a MySQL container and creates a bag of system call frequencies using a sliding window. Training is done by adding BoSC to a normal-behaviour database, and detection is done by comparing BoSC to that database, and if it does not exist, a mismatch has occurred. If the number of mismatches during a single epoch exceeds a certain threshold, an anomaly has been detected. Presents a detailed complexity analysis of the algorithm, which shows a relatively low complexity. However, it does not provide a concrete performance evaluation.
T3	S T I D E	Evaluates multiple classifier algorithms and the process of dimensional reduction using PCA. Dimensional reduction overall resulted in reduced processing time but decreased accuracy. C4.5 Decision Tree provides the best accuracy and lowest process time.
T4	S T E	Provides an extensive evaluation of an LSTM neural network with different configuration combinations applied to 7 types of attacks. Specifically targets a MySQL container with various modifications made to be able to execute the attacks. The model achieves ~90% accuracy overall. No performance evaluation is provided.
T5	D	Specifically created for Kubernetes. Trains a classification model using multiple algorithms and are compared using three separate datasets. Data is collected and stored using cAdvisor, Heapster and InfluxDB. Due to Heapster being created explicitly for Kubernetes, this technique is not relevant in our context. However, parts of the proposed technique might still be of interest.
T6	D	Proposes a solution that analyses the cause of increased resource utilisation, which causes an up-scaling of available resources. The solution aims to reduce infrastructure costs by diagnosing anomalous behaviour and decide what action is appropriate. The anomaly detection algorithm is a relatively simple outlier detection based on a modified Z-score and CPU usage. Anomaly identification is performed after an outlier has been detected, using decision trees. The technique is heavily tailored for clusters, which means that it is not relevant in our context.
T7	T I E	Shows that Sysdig and Falco can detect multiple misuse and attack scenarios using a rule-based approach. Performance evaluation shows that applying filters to what system calls to capture leads to significantly reduced overhead and disk usage.
T8	T	Technique has the potential to be applied to more generic cases as well. Decision Tree performs best of the tested algorithms in all regards.

ID	S T R I D E	Summary
T9	D	Provides a data preprocessing, labelling and anomaly detection technique. The preprocessing technique is to remove redundant data to reduce data dimensions. The data labelling technique identifies what resource deficiency is the cause of a container anomaly. The detection model is based on random forests, with ensemble learning using the K-means and kNN algorithms. The RF model is evaluated and compared to four other algorithms, where it performs the best (~85% accuracy). Evaluated in a cluster, but the technique applies to other cases. No mention of how data is collected and no performance evaluation provided.
T10	T I D E	Not a detection algorithm, but a framework that uses the Random Forest algorithm. The framework tries to overcome the challenge of lacking training data for short-lived containers. It enables real-time intrusion detection and is evaluated on a set of varying CVEs. Feature extraction performed by creating frequency vectors. Able to distinguish between different applications using application classification, meaning that input is first classified by which application it is from and then classified as an anomaly or not. This increases accuracy in a multi-tenant environment. CDL performs well at application classification but not as good at anomaly classification.
T11	D	Uses the component response time of every component in a multi-layered architecture to detect and localise anomalies. Able to identify CPU, memory and network anomalies. In order to monitor response times, every container needs to run an agent which collects it.
T12	D	Aims to detect and prevent economic denial of sustainability (eDoS) attacks by detecting malicious workloads and preventing automatic upscaling of microservices.
T13	D	Puts focus on the recovery aspect, which is not relevant in our context. Uses HHMM in order to detect and localise anomalies based on resource utilisation. Shows promising results (94% accuracy) and performs better than the Dynamic Bayesian Network and Hierarchical Temporal Memory detection techniques. It does not provide a performance evaluation.
T14	T D E	Uses a probabilistic technique on n-grams of system calls to determine the likelihood of a particular n-gram occurring. If the likelihood is below a threshold, an n-gram is flagged. If enough n-grams are flagged, the activity is considered malicious. The approach is evaluated on UNM datasets, using two different methods (MLE, SGT). MLE performs best when there are little training data, SGT otherwise. Average accuracy is ~90%. The performance evaluation shows that the CPU usage averages ~100% (1 core), which is relatively high.

ID	S T R I D E	Summary
T15	D	Provides multi-dimensional resource monitoring and log collection of clusters. The system is not especially security-oriented but focuses on data collection, workload prediction and rule-based alerts. Simple rules trigger alerts, e.g. "CPU utilisation is above 70% 100 times in 30 minutes". It should be regarded as more of a platform than a monitoring technique. It is not relevant in our context.
T16	T I D E	Tests a multitude of known vulnerabilities (CVEs) against multiple detection algorithms. Clair (static analysis tool) performs poorly compared to the other dynamic methods. The Self-Organising Map (SOM) yields the best TPR/FPR results but has a long detection time (~27s). The K-means algorithm provides good TPR/FPR results with a much better detection time (~0.36s). No performance evaluation of algorithms.
T17	D	Primary focus is on reducing the alarm delay, not detection accuracy. Describes a way of reducing the metrics dimension by identifying key metrics.
T18	I	Specifically tailored to detect Spectre and Meltdown attacks. Uses the perf command for low-level monitoring of the Linux system, which may not be relevant in our context. A non-intrusive method with a small performance overhead, meaning that the anomaly detection algorithm itself might be interesting.
T19	D	Proposes a fault injection framework to attack a system, deplete its resources, and propose several fault detection models based on quantile regression. Fault behaviour and interference phenomenons are observed to detect faults. The models are tested on four kinds of AI applications. The evaluation shows similar results for all applications with an average accuracy of ~93%. It does not provide a performance evaluation.
T20	D	Optimized iForest performs significantly better than iForest and noticeably better than LOF regarding TPR and FPR. The optimal number of iTrees is 100, yielding a computation time of ~40 ms. Rebuilds a new iForest around every 40 seconds, depending on the current (dynamic) monitoring period, meaning detection time is ~40 seconds. Thus, the optimised iForest is relatively performant.

4.1.2 Citation Matrix

Table 4.7 depicts what studies include references to other studies. In the matrix, an "X" denotes the row referencing the column. The "-" signifies a study not able to cite another study due to the publication dates.

T_x	1	2	12	19	17	3	5	14	9	16	20	11	6	13	7	4	15	8	18	10
1		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2			-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12				-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
19			-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
17			-	-		-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3	X						-	-	-	-	-	-	-	-	-	-	-	-	-	-
5								-	-	-	-	-	-	-	-	-	-	-	-	-
14		X							-	-	-	-	-	-	-	-	-	-	-	-
9							X			-	-	-	-	-	-	-	-	-	-	-
16											-	-	-	-	-	-	-	-	-	-
20												-	-	-	-	-	-	-	-	-
11			X	X	X								-	-	-	-	-	-	-	-
6			X				X					-		-	-	-	-	-	-	-
13					X										-	-	-	-	-	-
7	X															-	-	-	-	-
4	X						X										-	-	-	-
15					X													-	-	-
8	X	X																	-	-
18											X						-			-
10	X									X										

Table 4.7: Citation matrix, “X” denotes the row referencing the column, “-” signifies a similar or later publication date, making a reference unlikely.

4.2 Inter-rater Reliability Test

The inter-rater reliability test was performed using a starting set consisting of a single study [2] randomly selected from the actual starting set. The test was conducted two months after the literature review, leading to studies published after the review was conducted to be investigated during the test. Thus, to avoid inconsistencies, two recently published studies were removed from the test. The agreements and disagreements of inclusion and exclusion between the review and the test are presented in table 4.8. Calculating Cohen’s kappa using these values yields $\kappa = 0.843$, meaning *almost perfect agreement* according to the guidelines presented by Landis and Koch [34].

Review \ Test	Test	
	Include	Exclude
Include	3	0
Exclude	1	33

Table 4.8: Agreement and disagreement for inclusion and exclusion between the review and the test.

4.3 Literature Review Analysis

Early on in the process of extracting data from the included studies, three properties that compose most anomaly detection technique emerged. These are; **(I)** source of data analysed; **(II)** feature extraction algorithm; and **(III)** anomaly detection algorithm. It became apparent that the refined technique to be created in this thesis had to be composed in the same manner. However, limiting the refined technique to a single composition was deemed too restrictive as it would cause the performance of the refined technique to depend predominantly on the selected composition. Thus, given the lacking experience of the authors in this field, a decision was made to implement and evaluate multiple, uniquely composed techniques.

To reduce the number of unique techniques to be implemented and evaluated, the initial scope of potential compositions was narrowed down. It was accomplished by analysing the extracted data and understanding how existing techniques perform and are composed. The analysis led to the following conclusions regarding technique composition:

- (I)** System calls are the most common source of data analysed. It performs well and is, compared to resource utilisation, considerably superior from a STRIDE coverage perspective.
- (II)** Feature extraction can be performed in a plethora of ways, and the selection mainly depends on the choice of **(I)** and **(III)**.
- (III)** Machine learning algorithms are prevalent, perform well and are commonly paired with system calls as the type of data.

Based on these conclusions, the following decisions were made. Firstly, **system calls** was going to be the source of data that will be collected and analysed. A broader range of attacks, in regards to STRIDE, can be detected using system calls when compared to resource utilisation. The reason being that attempted attacks often cause changes to the behaviour of an application, which generally manifests itself through variations in the frequency of system calls invoked by the application [70]. Resource utilisation, on the other hand, is greatly influenced by the current workload and tends to be too noisy to be used for detecting attacks more sophisticated than a denial-of-service [70].

Secondly, it was decided that a **machine learning algorithm** would be implemented to identify anomalies in the collected system calls. The main reasoning being the prevalence of machine learning algorithms paired with system calls, which provides confidence in the composition, and promising results. However, deciding what algorithm to implement was not straightforward, as the novelty of the refined technique was of importance. The desire to implement and evaluate a novel technique, which has not yet been adequately evaluated in the literature, led to the *Optimised Isolation Forest* (T20) as presented by Zou et al. [82]. In their study, the authors present a modified Isolation Forest algorithm that has been optimised to detect anomalies in resource utilisation data. However, as the type of data chosen in this thesis is system calls, the original, non-optimised variant of the isolation

forest [44] would have to be implemented instead. Further research revealed only a single existing study evaluating the performance of isolation forest with n-gram feature extraction using system calls [45]. Thus, the decision was made to use the **Isolation Forest** algorithm for anomaly detection.

Lastly, it was decided to implement and evaluate multiple feature extraction algorithms. The decision was made based on the two previous choices. Given that the data type is system calls and that the anomaly detection algorithm was a machine-learning one, all possible candidates found in the extracted data were chosen. They were: **n-gram**, **frequency vector** and **sliding window**.

5

Implementation

This section presents the implemented container anomaly monitoring technique, including the different variations of the technique that were developed and evaluated during the iterative process of design science. First, the raw data source is explained and motivated, then the feature extraction algorithms that were evaluated are presented. Next, the anomaly detection model and the algorithm used to construct the model are presented. Lastly, the custom system call monitoring tool Heimdall and the anomaly detector evaluation framework Hlin developed to facilitate the iterative development process are presented. The implementation proposed in this chapter is developed for offline pre-processing and anomaly detection. Theoretically, the refined technique is capable of online anomaly detection, although it is not implemented nor evaluated as such in this thesis.

5.1 Data Source

Detecting anomalous behaviour of applications requires behavioural data to be collected. The data source chosen to train the model and identify anomalies is **system call traces**. System calls are used by processes running on a host to request services from the host OS kernel, e.g. reading/writing files or creating new processes. System calls acts as an abstraction layer that allows the kernel to provide controlled access to sensitive services without relying on the calling process behaving correctly. Because the OS kernel provides system calls, they can be traced without any modifications made to the monitored process itself, making it a non-intrusive, general-purpose monitoring method.

By analysing the system calls a process is invoking, patterns of normal behaviour emerges and can be collected to create a normal profile. The profile can then be used to analyse the current behaviour of a process to detect anomalous activity and potential attacks [22]. Forrest et al. [22] compare the approach to an immune system, capable of learning the definition of *self* and, by extension, what is not. Their study provides a simple method for defining *self* by using a short-sequence sliding window of a process' system calls and a method of comparing sequences of system calls to the definition of *self* to detect anomalous behaviour.

In order to trace and collect the system calls of a container, an existing tool is

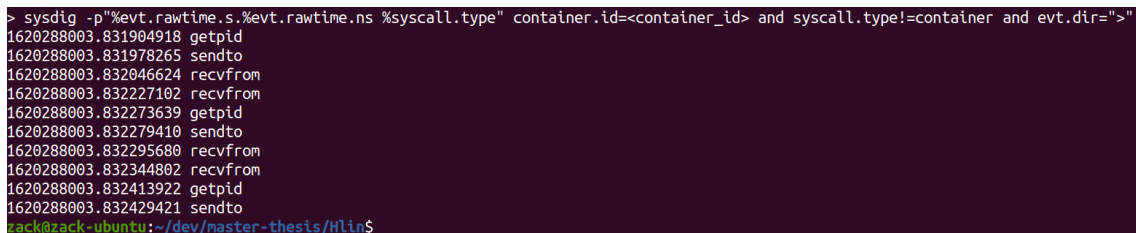
utilised. Sysdig [43] is a tool with native support for containers that provide *deep system visibility* by intercepting system calls and other OS events. The tool can be attached to a specific container and log the system calls invoked from the container, including timestamps, system call names, passed arguments and return values. An example of what Sysdig outputs is presented in figure 5.1. A custom system call monitoring tool, described in section 5.5.1, was developed as an extension to Sysdig, allowing the outputted system calls to be labelled. Having labelled data provides benefits over non-labelled data by enabling the use of classification-based machine learning algorithms, also called supervised learning, and the ability to produce confusion matrices in order to evaluate the performance of an anomaly detection model. Non-labelled data would restrict the machine learning solutions to unsupervised learning.

5.1.1 Alternative Sources

The behaviour of an application can manifest itself in different ways, providing options for potential attributes to monitor for anomalies. Different attributes provide different benefits and drawbacks regarding what types of anomalies can be detected and the level of intrusiveness required. Generally, the applicability of a monitoring approach increases the less intrusive it is, as more invasive approaches tend to become application-specific. For example, monitoring the internal state of an application, e.g. the number of failed login attempts or user activity, allows for detection of sophisticated attacks but requires unique modifications to the application itself, hence the increased intrusiveness and lower applicability.

System resource utilisation monitoring of a container, e.g. CPU utilisation or memory usage, is a non-intrusive approach that can detect anomalous application behaviour manifesting itself through anomalous resource utilisation, such as memory leaks or endless loops. The approach is excellent for detecting denial-of-service (DoS) attacks. However, it is limited to attacks that manifest themselves through resource utilisation and is unable to detect more sophisticated attacks.

Another approach is to let the applications themselves report their statuses, such as the time it takes to serve a request or the rate of specific requests. The approach can detect many types of attacks, as it can be tailored towards the specific application. However, this also makes it immensely intrusive as it requires changes to the monitored application's source code, making it a per-application monitoring



```
> sysdig -p"%evt.rawtime.s.%evt.rawtime.ns %syscall.type" container.id=<container_id> and syscall.type!=container and evt.dir=">"
1620288003.831904918 getpid
1620288003.831978265 sendto
1620288003.832046624 recvfrom
1620288003.832227102 recvfrom
1620288003.832273639 getpid
1620288003.832279410 sendto
1620288003.832295680 recvfrom
1620288003.832344802 recvfrom
1620288003.832413922 getpid
1620288003.832429421 sendto
zack@zack-ubuntu: ~/dev/master-thesis/Hlin$
```

Figure 5.1: Example of Sysdig output when attached to a container and logging enter events.

solution.

5.2 Data Representation

The data used to construct a machine learning algorithm needs to be pre-processed and encoded before being analysed. The encoding is often a numeric vector, commonly referred to as the **feature vector**. As the source data consists of sequences of system calls, which are categorical, encoding schemes capable of encoding categorical data into feature vectors are needed. Two types of encoding schemes have been employed in this implementation; Bag of System Calls and One-hot encoding.

5.2.1 Bag of System Calls

Bag of System Calls (BoSC) is a frequency-based feature vector first introduced by Kang et al. [29]. BoSC is a representation that removes the ordering information in the data and retains only the frequency information. The representation has shown promising results when used for anomaly detection at the process level [29].

Given an input sequence of system calls of length t :

$$\langle s_1, s_2, \dots, s_t \rangle$$

a Bag of System Calls is defined as:

$$\langle c_1, c_2, \dots, c_n \rangle$$

where n is the number of unique calls and c_i the number of occurrences of system call s_i in the input sequence. The size of a bag, s , is the number of inputs it was constructed from and is equal to the length of the input sequence, $s = t$. It is calculated as:

$$s = \sum_{i=1}^n c_i$$

Given the example output provided in figure 5.1, the following input sequence can be extracted:

$$\langle getpid, sendto, recvfrom, recvfrom, getpid, sendto, recvfrom, recvfrom, getpid, sendto \rangle$$

The input sequence has length $t = 10$ and $n = 3$ unique calls. Constructing a BoSC with $t = s = 10$ from the sequence results in the following bag:

$$\langle c_1, c_2, c_3 \rangle = \langle c_{getpid}, c_{sendto}, c_{recvfrom} \rangle = \langle 3, 3, 4 \rangle$$

BoSC can represent the same data in many different ways by changing how the source data is grouped into input sequences. For example, using the same input sequence described above, one could construct multiple BoSC by splitting the input sequence into sub-sequences of length s , effectively changing the bag size. Table 5.1 shows the resulting BoSC for different bag size values using the example input sequence.

s	Bags of System Calls
2	< 1, 1, 0 >, < 0, 0, 2 >, < 1, 1, 0 >, < 0, 0, 2 >, < 1, 1, 0 >
5	< 2, 1, 2 >, < 1, 2, 2 >
10	< 3, 3, 4 >

Table 5.1: Constructing BoSCs using $s = \{2, 5, 10\}$ from the example input sequence.

5.2.2 One-hot Encoding

One-hot encoding is a way of encoding categorical data into feature vectors while retaining ordering information. In contrast to label encoding, which assigns each categorical value with an integer, one-hot encoding uses boolean values to represent categories. One-hot encoding thus brings an advantage over label encoding as the latter introduces an implicit ordering of the categorical values. For example, a label encoder could assign categories A, B and C the values 1, 2 and 4, respectively. Due to the inherent ordering of these numerical values, an arbitrary relationship between the categories is introduced that the machine learning algorithm will try to understand. The algorithm could interpret this encoding as if categories A and B are more similar compared to category C, even though the categories have no relationship.

One-hot encoding represents a single categorical value using multiple boolean values. The number of boolean values needed to represent a single categorical value is equal to the number of unique system calls. Only one of these boolean values must be true, indicating the categorical value of that column in the feature vector. Given an input of three n-grams of system calls to encode:

< getpid, sendto, recvfrom >, < recvfrom, getpid, sendto >, < sendto, recvfrom, getpid >

The one-hot encoder retrieves all possible values for the three columns and represents them using boolean values (0/1). In this case, *getpid* is represented by the first boolean value, *sendto* by the second one and *recvfrom* by the third one. The resulting one-hot encoding is the following matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

5.3 Feature Extraction

Feature extraction is a vital part of machine learning. It is the process of transforming the data into a representation better suited for training a machine learning algorithm. Commonly, feature extraction involves dimensional reduction and encoding the data as a feature vector. Dimensional reduction is the process of removing redundant or irrelevant information from the source data and often results in lower

computational complexity, due to fewer features, and increased accuracy, due to increased relevancy of features. During the iterative design science process, two encoding types (described in section 5.2) and three feature extraction algorithms were implemented to be evaluated and compared:

- Frequency vector
- Sliding window
- n-gram

All implemented feature extraction algorithms have used the same approach to label the feature vectors that they output. If a feature vector has been created from a sequence of system calls where at least one of the system calls is labelled as anomalous, the entire feature vector is labelled as anomalous.

5.3.1 Frequency Vector

The frequency vector feature extraction algorithm is based on frequency vectors as described by Tunde-Onadele et al. [70]. Frequency vectors are BoSC where the input sequence has been split into sub-sequences of equal *time intervals*. By grouping the input based on time, the data representation partly encodes the current load of the monitored application, making it the only feature extraction algorithm included in this research capable of doing so.

Constructing frequency vectors requires that the input sequence include timestamps for each input. Using these timestamps, time frames can be constructed by which the inputs are grouped. A time frame is defined as an interval in milliseconds:

$$[T, T + \Delta T)$$

where $\Delta T = 100ms$, as proposed by Tunde-Onadele et al. [70]. Given an input sequence of length n , where each input is a pair of a system call and a millisecond timestamp:

$$\langle i_1, i_2, \dots, i_n \rangle$$

$$i_x = \langle s_x, t_x \rangle$$

which occurred over a time period equal to or greater than ΔT ($t_n - t_1 \geq \Delta T$), multiple time frames can be constructed:

$$\langle [T_1, T_2), [T_2, T_3), \dots, [T_{m-1}, T_m) \rangle$$

$$T_i - T_{i-1} = \Delta T$$

where $T_1 = t_1$ and $t_n \in [T_{m-1}, T_m)$.

Using the example input sequence defined previously in section 5.2.1 with $t_1 = 1620377398948ms$ and subsequent inputs occurring 30 milliseconds apart, the following input sequence is created:

$$\langle \langle getpid, 1620377398948 \rangle, \langle sendto, 1620377398978 \rangle, \langle recvfrom, 1620377399008 \rangle, \dots \rangle$$

Using $\Delta T = 100ms$, the following time frames would be constructed:

$$T_1 = 1620377398948 \quad T_2 = 1620377399048$$

$$T_3 = 1620377399148 \quad T_4 = 1620377399248$$

The results of grouping the example input using these time frames and creating BoSC from the groups are presented in table 5.2. It shows how the bag sizes can differ between individual BoSC, which is how the current load of the monitored application is encoded.

System call Time frame	getpid	sendto	recvfrom	BoSC
$[T_1, T_2)$	1	1	2	$\langle 1, 1, 2 \rangle$
$[T_2, T_3)$	1	1	1	$\langle 1, 1, 1 \rangle$
$[T_3, T_4)$	1	1	1	$\langle 1, 1, 1 \rangle$

Table 5.2: Creating BoSC from frequency vectors using the example input sequence with $\Delta T = 100ms$.

5.3.2 Sliding Window

The sliding window method of feature extraction groups the source data by sliding a window over it, extracting an input sequence every time a step is taken. The extracted input sequences are then used to create BoSC. A visualisation of the algorithm is presented in figure 5.2.

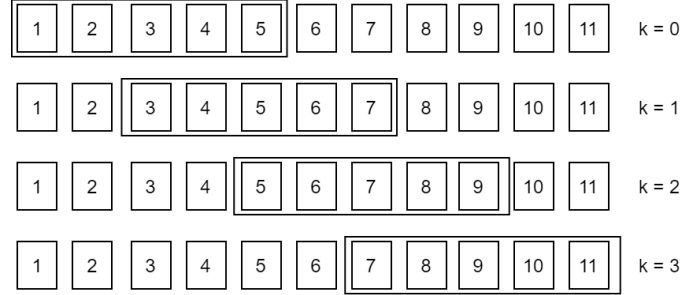


Figure 5.2: Visualization of the sliding window algorithm using a trace of $t = 11$ system calls, window size $W = 5$ and step size $L = 2$.

The sliding window algorithm takes two parameters; the window size, W , and stepping size, L . Given an input sequence of system calls with length t :

$$\langle s_1, s_2, \dots, s_t \rangle$$

A sliding window with window size W and stepping size L would extract the following input sequence at step k :

$$[s_{k*L+1}, s_{k*L+W}] \quad k \in \mathbb{N} \cap [0, \lceil \frac{t-W}{L} \rceil]$$

The extracted input sequences are of length W , which means that all the BoSC produced are of size W as well. For cases where the upper bound value of k is not an integer, it is rounded up to the nearest integer. In other words, if the original input sequence results in the last bag having a size less than W , it is still included.

As an example, for $t = 11$, $W = 5$ and $L = 2$, the following input sequences would be extracted, as illustrated in figure 5.2:

$$< [s_1, s_5], [s_3, s_7], [s_5, s_9], [s_7, s_{11}] >$$

As the implementation is based on offline data analysis, a scenario where there is not enough system calls in the sequence to produce a whole number of bags of the same size might occur. In this scenario, the algorithm outputs smaller BoSC instead of ignoring the last system calls in the input sequence. Using the example input sequence defined in section 5.2.1 ($t = 10$), the algorithm with $W = 5$ and $L = 2$ would produce the BoSC listed in table 5.3. The table illustrates how the last bag produced by the algorithm is smaller than all other bags, which can be confirmed by calculating that the upper bound (without rounding upwards) is not an integer:

$$\frac{t - W}{L} = \frac{10 - 5}{2} = 2.5$$

Step (k)	Extracted input sequence	Bag of System Calls
0	$[s_1, s_5]$	$< 2, 1, 2 >$
1	$[s_3, s_7]$	$< 1, 1, 3 >$
2	$[s_5, s_9]$	$< 2, 1, 2 >$
3	$[s_7, s_{10}]$	$< 1, 1, 2 >$

Table 5.3: Using a sliding window with $W = 5$ and $L = 2$ to construct BoSC from the example input sequence.

5.3.3 n-gram

The n-gram technique is, in contrast to previously explained algorithms, a more straightforward feature extraction approach. N-grams are created by splitting the input sequence into sub-sequences of length N . These are also referred to as *non-overlapping* n-grams. The extracted feature vectors thus keep the original ordering, which the previous algorithms do not. The algorithm is a special case of the sliding window algorithm (section 5.3.2), where the window size is equal to the stepping size, but the output is not BoSC.

Producing n-grams of length N given an input sequence of length t :

$$< s_1, s_2, \dots, s_t >$$

is achieved by extracting the following interval at step k from the input:

$$[s_{k*N+1}, s_{k*N+N}] \quad k \in \mathbb{N} \cap [0, \lceil \frac{t}{N} - 1 \rceil]$$

For cases where the upper bound value of k is not an integer, it is rounded up to the nearest integer. In other words, if the original input sequence results in the last n -gram having a size less than N , it is still included.

Using the example input sequence as presented in section 5.2.1 and $N = 5$, the following two n -grams are produced:

$$< [s_1, s_5], [s_6, s_{10}] >$$

$$< getpid, sendto, recvfrom, recvfrom, getpid >, < sendto, recvfrom, recvfrom, getpid, sendto >$$

The output of the n -gram algorithm does, however, need to be encoded into a numerical vector representation that a machine learning algorithm can take as input. To this end, one-hot encoding (section 5.2.2) has been used in favour of BoSC as the former retains the ordering information while the latter removes it.

5.4 Anomaly Detection

Detecting anomalies in the data sets created by the feature extraction algorithms is achieved by employing a machine learning algorithm. Machine learning algorithms are commonly divided into two types, supervised and unsupervised, with the main difference between the types being the use of labelled data. The data used in this thesis have been labelled, enabling the use of either type of algorithm. However, only unsupervised algorithms have been implemented in this thesis.

Two algorithms have been implemented for anomaly detection purposes in this thesis, the k -nearest neighbours (kNN) and the Isolation Forest (iForest) algorithm. The kNN algorithm is employed in the baseline technique, which has been implemented in order to establish a frame of reference in which the isolation forest can be evaluated and compared. The baseline technique that implements the kNN algorithm is based on the works of Tunde-Onadele et al. [70].

5.4.1 Baseline Technique: k -nearest neighbours

The baseline technique [70] utilises the frequency vector feature extraction algorithm (section 5.3.1) and the kNN algorithm for anomaly detection. Generally, the kNN algorithm is used as a supervised algorithm, but it can also be used as an unsupervised algorithm, with the latter being the case in this case. The unsupervised algorithm is a relatively simple algorithm that, given a data set, calculates the distances for each point in the data set to its k nearest neighbours. It uses the euclidean distance metric to calculate the distances. The algorithm has two parameters:

- k - Number of nearest neighbours to consider

- p - Anomaly threshold percentile

The algorithm considers a data point anomalous if the average distance to its k nearest neighbours is in the top p percentile of all data points in the data set. The parameters used to evaluate the baseline is $k = 5$ and $p = 10\%$, as empirically found to be optimal [70].

The implementation uses scikit-learn's NearestNeighbors¹ module, which provides the unsupervised algorithm for creating the model and calculating the distances between a data point and its k nearest neighbours. All parameters except $n_neighbors$ are set to their default values.

5.4.2 Refined Technique: Isolation forest

The refined technique performs anomaly detection using the iForest algorithm, inspired by the *Optimised Isolation Forest* as proposed by Zou et al. [82]. However, the optimisations presented by the authors are specific to resource utilisation and does not translate when using system calls as the data source. Thus, the original iForest algorithm is used instead.

The iForest algorithm has multiple beneficial characteristics that influenced the decision to evaluate its performance utilising system calls.

- It has low linear time complexity and a small memory requirement.
- It deals well with high dimensional data with irrelevant attributes.
- It allows for training without anomalous data as it is an unsupervised algorithm; training in this context means a definition of normal.
- It can provide detection results with different levels of granularity without re-training [11].

The iForest consists of several isolation trees (iTree) where each leaf node is a single data. The base premises are as follows; data easy to isolate means it is in a sparse part of the data set, an outlier, is likely to be an anomaly. The sooner the algorithm manages to isolate data from the set, the more likely it is an anomaly. Constructing an iTree using a set of N data items follows the following steps;

1. First, take n samples from the set (N).
2. Then, randomly select a feature and a p -value within the range of all values this feature has in the set. This p -value will be placed at the root of a binary tree. All data items whose feature value is less than p is placed on the left side, others on the right.
3. repeat previous steps on the left and right data items until the data is divisible no more or if the tree's height reaches $\log_2(n)$.

Repeating the described process of generating iTrees constitutes the creation of an

¹<https://scikit-learn.org/0.24/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors>

iForest. Structuring the data in this way means there is a distance, $h(x)$, between each data item, x , and the root. The average of all $h(x)$ is denoted $E(h(x))$. The anomaly value of data x in the n samples is denoted $S(x, n)$. where the anomaly value is calculated as follows:

$$S(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (5.1)$$

$$c(n) = 2H(n-1) - \left(\frac{2(n-1)}{n}\right), H(i) = \ln(i) + e \quad (5.2)$$

The range of $S(x, n)$ is $[0, 1]$, where values closer to 1 constitutes a high probability of an anomaly and values close to 0 means a high probability of normal behaviour. If most values are close to 0.5, the entire data set is considered to have no apparent outliers.

The implementation of the iForest algorithm in this research has two adjustable parameters, the number of estimators (iTrees) and the contamination (cont) percentage of the data. The contamination is the proportion of outliers in the data set. If no value is given for the contamination rate, the threshold is instead determined as in the original paper [44]. During evaluation, we refer to this setting as *auto*. The *exact* setting is the calculated percentage of anomalous BoSC's or N-grams from the labelled data, and the percentage values refer to the *exact* value with an upper and lower bound, using the limits as input if the exact value falls outside the defined range.

The implementation uses the IsolationForest² module provided by scikit-learn to create the model from and perform predictions on the data set. All parameters except *n_estimators*, *contamination* and *random_state* are set to their respective default values.

5.5 Simulation Environment Tools

This section presents Heimdall, a tool developed to collect and label container behavioural data through system calls, and Hlin, a framework developed to facilitate the evaluation of anomaly detection techniques. The two tools are used to perform the evaluation of the refined techniques implemented in this thesis.

5.5.1 Heimdall - System Call Monitoring Tool

System calls need to be monitored and logged to create training data sets for the anomaly detection algorithms. We created Heimdall³ to manage monitoring, logging

²<https://scikit-learn.org/0.24/modules/generated/sklearn.ensemble.IsolationForest.html>

³<https://github.com/ZLundqvist/Heimdall>

and labelling of system calls. Heimdall is written in TypeScript and runs in the Node.js runtime. It is an extension of the existing tool Sysdig [43] which provides *deep system visibility* by intercepting system calls and other OS events with native support for monitoring containers. Compared to existing tools, Heimdall expedites the process of monitoring and logging system calls of victim containers by providing the following additional functionalities:

- Automatically attach Sysdig to containers as they start
- Redirect output of Sysdig to a per-container log file
- Labelling of output data
- RESTful API to toggle labelling mode of container

An overview of the architecture of Heimdall can be seen in figure 5.3. The tool automatically attaches to containers when they start by listening to events emitted by the Docker Engine API. Once the tool detects a container-start event, a new Sysdig process is started by executing the command:

```
sysdig -p"%evt.rawtime.s.%evt.rawtime.ns %syscall.type" \  
    container.id=<container_id> \  
    and syscall.type!=container \  
    and evt.dir=">"
```

The command instructs Sysdig to output system calls made by the started container in a format that only includes the timestamp and the system call name. The tool captures the output of every Sysdig process, labels it, and redirects it to a per-container log file.

The labelling of output data is done by prepending a flag to each output of Sysdig, representing either normal or anomalous activity. The tool's RESTful API provides endpoints to toggle a container's current labelling mode, simplifying the data labelling process into a single HTTP request sent to the monitoring agent at the start

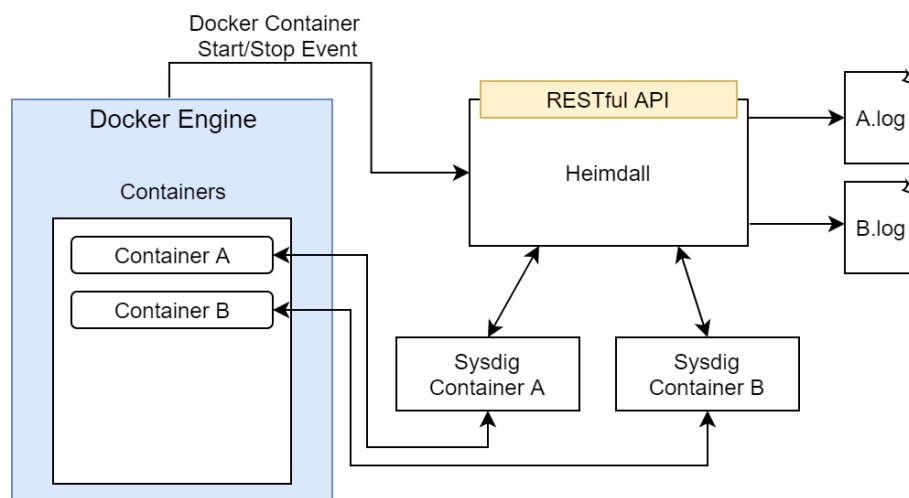


Figure 5.3: Architecture of Heimdall.

and end of an attack providing correctly labelled data as output.

5.5.2 Hlin - Anomaly Detector Evaluation Framework

The need for a framework to aid in developing a refined technique was discovered early in the ideation phase. It was realised that multiple feature extraction algorithms, anomaly detection algorithms, and combinations were to be evaluated. A framework would expedite the process of evaluating new or modified components. It would handle integrating the components and allow us to focus solely on implementing the components themselves, thus speeding up the iterative process of refining a technique.

The framework developed was given the name Hlin⁴, and its architecture is presented in figure 5.4. It uses a three-layered architecture, including a feature extraction layer, transform layer and anomaly detection layer. The feature extraction layer converts the raw system call traces to a generic representation of the data stored in a pandas DataFrame⁵, with generic referring to the data still having to be transformed into a suitable format for the selected algorithm. DataFrames are cached in order to skip the feature extraction step for subsequent runs. The transform layer consists of a single transformation algorithm for each combination of feature extraction and anomaly detection algorithm. The transform layer transforms generic DataFrames output by the feature extraction layer to the format expected by the selected anomaly detection algorithm. The transformations are relatively simple and include, for instance, memory optimisations and the data contamination calculation for the isolation forest. The anomaly detection layer is responsible for constructing and validating an anomaly detection model and outputting the confusion matrix produced from the external validation. The architecture enables the framework to provide easy integration of components and fast implementation, iteration and evaluation of feature extraction and anomaly detection algorithms.

The framework is written in Python and utilises the well-established scikit-learn library [52, 63] for most machine learning-related tasks, i.e. the anomaly detection

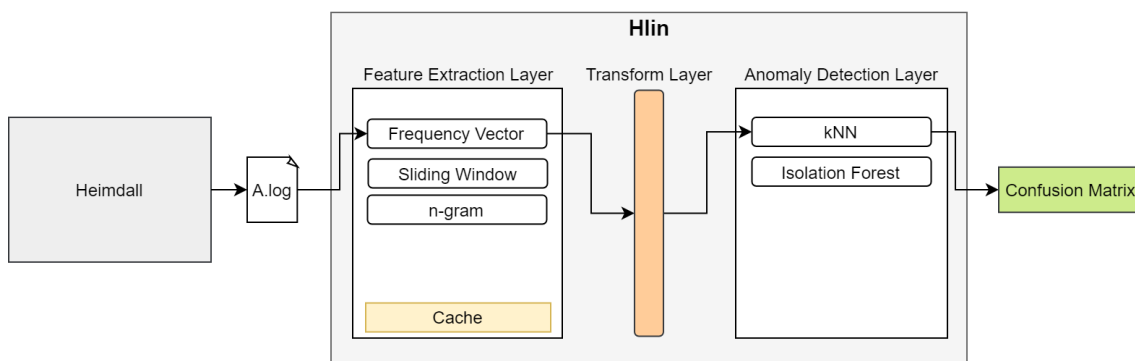


Figure 5.4: Architecture of the Hlin framework.

⁴<https://github.com/ZLundqvist/Hlin>

⁵https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframe

layer components as described in section 5.4. The framework takes the following arguments as input:

- Input file or directory of input files as produced by Heimdall, the custom system call monitoring tool.
- Feature extraction component (frequency vector, sliding window, n-gram) to use.
- Anomaly detection component (knn, isolation forest) to use.
- Parameters required by the selected feature extraction and anomaly detection components (e.g. window size for sliding window)

The output of the framework is a csv file with metadata of the evaluation and the confusion matrix produced by external validation of the model.

6

Evaluation

This chapter first describes how the simulation environment was set up to evaluate the many technique compositions and parameter combinations of this thesis. After that, the iterative process of refining a monitoring technique by tuning parameters is detailed. Lastly, the refined monitoring technique is presented and compared to the baseline technique.

6.1 Simulation Environment

The simulation environment was designed to provide a consistent and reproducible environment in which containerised vulnerable applications can be run, monitored and exploited. Interestingly, consistency and reproducibility are intrinsic properties of Docker containers which simplifies the development of the environment. Specifically, we need a simulation environment designed to perform the following tasks:

- (I) Create Docker containers running applications with known exploits.
- (II) Collect system call logs.
- (III) Generate normal load for containerised applications.
- (IV) Exploit containerised applications.

The hardware and software configurations of the system where the evaluation is conducted are presented in tables 6.1 and 6.2, respectively. The system runs the victim containers, the exploits and the previously described custom monitoring tool and framework. The rest of this section describes how the simulation environment was set up to perform the above tasks.

Component	Specification
Processor	Intel i7 6700K 4.0 GHz
Memory	32GB DDR4 3000MHz
Storage	Samsung 980 Pro SSD

Table 6.1: Hardware configuration.

Software	Version
Operating System	Ubuntu 20.04 LTS
Docker	20.10.6 build 370c289
Docker-compose	1.28.6 build 5db8d86fd
Sysdig	0.27.1
Node.js	14.16.1
Python	3.8.5

Table 6.2: Software configuration.

6.1.1 Creating Vulnerable Docker Containers

To get a holistic view of how a monitoring technique performs, a diverse set of commonly containerised applications with known exploits has been created to be monitored and are listed in table 6.3. The *Threat Impact* column describes the type of attack the application is vulnerable to. The list of Common Vulnerabilities and Exploits (CVE) is managed by the CVE Numbering Authorities (CNA), where the *CVE ID* is the common way of identification. The *CVSS column* lists the CVSS score, the degree of severity and characteristics of the exploit. The following two columns list the vulnerable application and its version. The second to last column, *Exploit Tool*, lists the tool used to perform the exploits, with *POC* referring to the use of Proof of Concept code. Finally, the *STRIDE* column refers to what category of STRIDE the attack actually targeted. This column does not directly correlate to the first column since CVE's belonging to the two first categories of threat impact allows a choice of what code to execute. With the intent of keeping the footprint of each attack small, only one or two commands were executed for each container. Finding suitable applications to include was achieved by looking at similar lists found in papers from the conducted literature review [41, 70], online collections of pre-built vulnerable Docker containers [73] and the Common Vulnerabilities and Exposures (CVE) database provided by NIST [50].

Threat Impact	CVE ID	CVSS Score	Application	Version	Exploit Tool	STRIDE
Return Shell and Execute Arbitrary Code	CVE-2014-3120	6.8	Elasticsearch	1.1.1	Metasploit	IE
	CVE-2015-3306	10	proFTPD	1.3.5	POC	IE
	CVE-2017-7494	10	Samba	4.5.9	POC	TE
	CVE-2016-3714	10	ImageMagick	6.7.9	POC	TIE
	CVE-2015-8562	7.5	Joomla	3.4.5	POC	IE
Execute Arbitrary Code	CVE-2017-12615	6.8	Apache Tomcat	8.5.19	POC	T
	CVE-2014-6271	10	Bash	4.2.37	POC	IE
	CVE-2015-1427	7.5	Elasticsearch	1.4.2	POC	IE
	CVE-2018-16509	9.3	Ghostsript	9.23	POC	IE
	CVE-2018-19475	6.8	Ghostsript	9.25	POC	IE
	CVE-2019-6116	6.8	Ghostsript	9.26	POC	IE
	CVE-2016-10033	7.5	PHPMailer	5.2.17	POC	T
	CVE-2017-5638	10	Struts	2.5	POC	I
Escalation of Privilege	CVE-2017-12635	10	CouchDB	2.1.0	POC	E
Disclose Credential Information	CVE-2015-5531	5	Elasticsearch	1.6.0	POC	I
	CVE-2016-1897	4.3	ffmpeg	2.8.4	POC	I
	CVE-2017-8917	7.5	Joomla	3.7.0	SQLmap	I
	CVE-2017-7529	5	Nginx	1.13.2-1	POC	I
	CVE-2017-5223	2.1	PHPMailer	5.2.20	POC	I
Crash the Application	CVE-2015-5477	7.8	Bind	9	POC	D
	CVE-2016-7434	4.3	NTP	1.4.2.8	POC	D
Consume Excessive CPU	CVE-2014-0050	7.5	Apache Commons FileUpload	1.3.1	POC	D

Table 6.3: Vulnerable containers used in the simulation environment.

6.1.2 Collecting System Call Logs

System call logs for the containers were collected using Heimdall, described in section 5.5.1. Logs were collected by first starting Heimdall and then starting a container. After starting the container, all invoked system calls are logged until manual termination or termination due to crash. The procedure ensures that *all* invoked system calls are logged during the entire lifetime of the container. Logs for all containers were collected before proceeding with the evaluation of the detection techniques.

6.1.3 Normal Load Generation

Generating a normal load is an essential aspect of creating a data set of good quality. Ideally, a normal load should include every scenario and type of load one would expect to see during real-world operation. However, the simulation environment includes different types of applications of varying complexities, which has forced the scope of generating normal loads to be reduced. For applications with low complexity that only perform a single or a few tasks, generating a normal load is straightforward. For more complex applications, however, only the most common scenarios have been simulated. Multiple tools were employed to generate the normal load, some of which have been used to generate a normal load for multiple applications. Many of the created applications use a web server to expose an interface used to interact with the vulnerable application. An excellent tool for load-testing web applications is the Apache JMeter application, which has been employed to generate a normal load whenever possible. JMeter’s HTTP sampler was used to send HTTP requests to web server applications such as Nginx, Joomla, PHPMailer, Apache and Struts. Normal behaviour data for proFTPD and samba were generated by uploading, downloading, replacing and deleting files using the *FTP sampler* and *OS process sampler*, respectively. ImageMagick and Ghostscript data were generated by sending images via HTTP POST requests. NTP and Bind received UDP requests using the JMeter *UDP request plugin* for the current time and DNS-lookups, respectively. Elasticsearch and CouchDB normal behaviour were replicated by adding, removing and searching for entries.

Containerised applications typically run for a short period of time, making it challenging for an anomaly detection system to collect sufficient data to build a complete model of normal behaviour. With the intent of reflecting these short lifespans, the data collection used for evaluation is based on six minutes of runtime. While running, the synthetic normal load is applied, and the attack is executed once. The duration was inspired by Tunde-Onadele et al. [70], who used a similar approach to evaluate various anomaly detection algorithms for containerised applications.

6.1.4 Exploitation

The majority of the exploits used against the vulnerable applications has been implemented as proofs-of-concept, as seen in table 6.3. Even though tools exist that can be used to expedite vulnerability testing, e.g. Metasploit, not much use was found for them as they rarely succeeded in exploiting the target applications. It was found

that implementing proof-of-concept exploits was the faster approach, especially considering the online resources available for many CVEs. Whether it was caused by the specific setup of the simulation environment or the authors' inexperience using these tools is unknown. The process of executing an exploit is a three-step process. First, an HTTP request is sent to the monitoring agent, telling it to switch to the anomaly labelling mode. The attack is executed, and lastly, another HTTP request is sent to revert the labelling mode of the monitoring agent.

Using the three-step process in order to label the collected system calls does not result in *perfectly labelled data*, which refer to data where only the system calls invoked as a result of the malicious action itself would be labelled as anomalous. Labelling data using that approach would be infeasible, if not impossible, but is also not necessary. This thesis intends to evaluate how well an anomaly detection technique performs in detecting anomalous activity within a *stream of activity*. Thus, the three-step process is a reasonable solution as it effectively marks the start and end of the malicious activity in the stream. The time frame of the malicious activity can then be used to validate whether models are capable of correctly identifying any anomalous activity.

6.2 Evaluation Metrics

The metrics used to present and compare the performance of the evaluated techniques are derived from confusion matrices, which are a common way of visualising the performance of classification algorithms [53]. Many performance-representing values can be derived from a confusion matrix, but only a few are relevant for this thesis' purpose. Specifically, the True Positive rate (TPR) and False Positive rate (FPR) are of interest.

The FPR is of interest due to the nature of the problem addressed in this thesis. How often a monitoring technique incorrectly classifies non-anomalous behaviour as anomalous, also called false alarms, is synonymous with the FPR, making it a valuable metric. Table 6.4 presents how the FPR is calculated. The TPR is a valuable metric due to how the simulation has been conducted. Because only a single attack is executed during the entire lifespan of a container, the TPR can be used to derive the binary value of whether a monitoring technique detects the attack or not. The proportion of correctly identified anomalous data points is not of interest, as the *extent* to which a monitoring technique can detect an attack is irrelevant. Thus, if the TPR of a monitoring technique is $> 0\%$, the monitoring technique has successfully detected the attack.

6.3 Iteration 0 (baseline)

Before refining a technique, the baseline technique has to be evaluated using the same data set as the refined technique to constitute a valid comparison. The baseline acts as a frame of reference for the refined techniques and is used for the comparison of

Metric	Description	Calculation
True Negatives (TN)	Correct negative classifications	-
False Positives (FP)	Incorrect positive classifications	-
False Positive rate	Proportion of incorrectly classified negatives	$\frac{FP}{FP+TN}$

Table 6.4: Explanations of evaluation metrics.

the evaluations. The baseline technique is implemented as previously described in section 5.4.1.

The evaluation results is presented in detail in table 6.5. The table shows that the baseline technique is able to detect 18/22 attacks with an average FPR of 7.37%, which is considerably better than the evaluation of the same technique performed by Tunde-Onadele et al. [70] (9/28, 9.92%). The discrepancy might be explained by the evaluations being performed on different data sets, reinforcing the importance of performing a baseline evaluation on the data set.

Threat Impact	CVE ID	Detected	FPR	STRIDE
Return Shell and Execute Arbitrary Code	CVE-2014-3120	✗	1.41%	IE
	CVE-2015-3306	✗	2.45%	IE
	CVE-2017-7494	✗	0.09%	TE
	CVE-2016-3714	✓	1.34%	TIE
	CVE-2015-8562	✓	0.24%	IE
Execute Arbitrary Code	CVE-2017-12615	✓	9.94%	T
	CVE-2014-6271	✓	9.91%	IE
	CVE-2015-1427	✓	9.80%	IE
	CVE-2018-16509	✓	9.99%	IE
	CVE-2018-19475	✓	9.98%	IE
	CVE-2019-6116	✓	9.97%	IE
	CVE-2016-10033	✗	10.03%	T
	CVE-2017-5638	✓	9.40%	I
Escalation of Privilege	CVE-2017-12635	✓	0.92%	E
Disclose Credential Information	CVE-2015-5531	✓	10.20%	I
	CVE-2016-1897	✓	7.99%	I
	CVE-2017-8917	✓	9.98%	I
	CVE-2017-7529	✓	16.02%	I
	CVE-2017-5223	✓	9.48%	I
Crash the Application	CVE-2015-5477	✓	10.00%	D
	CVE-2016-7434	✓	3.10%	D
Consume Excessive CPU	CVE-2014-0050	✓	9.87%	D
Average		81.82%	7.37%	

Table 6.5: Baseline technique evaluation results.

6.4 Iteration 1

The first iteration was conducted using a *shotgun* approach, meaning that the goal was to evaluate the performance of all technique compositions, as described in section 4.3, using only a few combinations of parameters. This approach yields a rough estimate of how the different compositions compare to each other, which is a good starting point. The evaluated compositions include the iForest algorithm for anomaly detection and all feature extraction algorithms.

6.4.1 Iteration 1: Parameters

The feature extraction and anomaly detection algorithms, including the parameter combinations used in the first iteration, are listed in table 6.6. The values for the number of iTrees is inspired by Zou et al. [82], who concluded that 100 iTrees yielded the best performance in their research. In the interest of exploring further, two more values are added.

The number of unique combinations to be evaluated for this iteration is the number of unique anomaly detection algorithms multiplied by the number of unique feature extraction algorithms, meaning $12 * 11 = 132$ unique techniques will be evaluated.

iForest		Feature Extraction		
Parameters	Values	Algorithm	Parameters	Values
iTrees	50, 100, 200	n-gram	n	3, 5, 10
Contamination	auto, exact, 1-5%, 1-7%	Sliding Window	(W, L)	(7,4), (10,5), (11,6), (11,11), (23,12)
		Frequency Vector	ΔT [ms]	50, 100, 200

Table 6.6: Anomaly detection and feature extraction parameters for the first iteration.

6.4.2 Iteration 1: Evaluation

The evaluation results have been split into two tables, where table 6.7 lists a selection of results from the sliding window feature extraction algorithm, and table 6.8 lists the results of combinations using the n-gram and frequency vector feature extraction algorithms. Given the amount of information, the results presented for this and future iterations is selected based on what conclusions came from the iteration in question. The results for this iteration showed a clear divide in terms of performance depending on which contamination mode was employed, which is why the five best performing combinations using each contamination mode (*exact*, *auto* and *custom*) is included in the reported results.

The first iteration made it clear that the *exact* contamination mode tends to be too conservative, detecting 16 out of 22 attacks at best. The *auto* contamination mode is the only mode that manages to detect all the attacks in this iteration but comes at the cost of a higher FPR ($\sim 13\%$). Using the 1-5% contamination interval and a sliding window with $W = 11$, $L = 6$ and 50 iTrees, the technique manages to detect

Sliding Window					
Cont	iTrees	Window Size	Step Size	Detected	FPR
Auto	50	11	6	100.00%	7.31%
1-7%	50	11	6	95.45%	2.20%
Auto	200	10	5	95.45%	6.55%
Auto	200	11	11	95.45%	6.74%
Auto	100	10	5	95.45%	6.76%
Auto	50	10	5	95.45%	6.98%
1-5%	200	10	5	90.91%	1.82%
1-5%	200	11	6	90.91%	1.83%
1-5%	50	11	11	90.91%	1.85%
1-5%	50	10	5	90.91%	1.86%
Exact	100	23	12	72.73%	3.52%
Exact	200	11	6	72.73%	3.55%
Exact	50	23	12	68.18%	3.51%
Exact	100	11	6	68.18%	3.62%
Exact	200	10	5	68.18%	3.65%

Table 6.7: Evaluation results first iteration: sliding window feature extraction.

N-gram					Frequency Vector				
Cont	iTrees	N-gram size	Detected	FPR	Cont	iTrees	ΔT [ms]	Detected	FPR
1-5%	50	5	90.91%	1.73%	Auto	200	50	100.00%	12.72%
1-7%	50	5	90.91%	2.16%	Auto	200	200	100.00%	12.83%
1-5%	200	3	86.36%	1.56%	Auto	50	200	100.00%	12.95%
1-5%	100	3	86.36%	1.58%	Auto	50	50	100.00%	12.98%
1-5%	100	10	86.36%	1.75%	Auto	100	200	100.00%	13.07%
Exact	100	5	72.73%	3.51%	1-5%	50	50	86.36%	2.17%
Exact	200	3	68.18%	3.35%	1-5%	200	50	86.36%	2.17%
Exact	50	10	63.64%	3.55%	1-7%	200	50	86.36%	2.62%
Exact	100	3	59.09%	3.43%	1-7%	50	50	86.36%	2.64%
Exact	200	5	59.09%	3.50%	1-7%	100	50	86.36%	2.64%
Auto	50	5	18.18%	1.38%	Exact	50	100	50.00%	5.17%
Auto	200	3	18.18%	1.64%	Exact	200	50	50.00%	5.19%
Auto	100	3	18.18%	1.85%	Exact	100	50	50.00%	5.20%
Auto	50	3	18.18%	2.16%	Exact	50	50	50.00%	5.22%
Auto	100	10	13.64%	0.82%	Exact	200	100	45.45%	5.19%

Table 6.8: Evaluation results first iteration: n-gram and frequency vector feature extraction.

21 attacks and has a relatively low FPR (1.87%). For future iterations, the *auto* and *exact* contamination modes are omitted in favour of using set intervals due to its superior overall performance.

Regarding the feature extraction algorithms, frequency vector paired with a contamination interval seems to detect slightly fewer attacks and result in a higher

FPR when compared to the other two algorithms. The only frequency vectors that perform well do so in combination with the *auto* contamination mode. From this iteration, it can be concluded that frequency vectors seem to perform worse than the other two feature extraction algorithms when paired with contamination intervals and is therefore excluded in subsequent iterations.

6.5 Iteration 2

The first iteration indicated that larger values of n for the n -gram feature extraction algorithm led to worse results. With the intent of confirming this, a larger n -value is evaluated for this iteration.

6.5.1 Iteration 2: Parameters

This iteration is considerably more focused than the previous. A total of 24 results are presented, six of which are *new* for this iteration, caused by the addition of $n = 20$. The complete set of parameter values evaluated as part of this iterated are listed in table 6.9.

iForest		Feature Extraction		
Parameters	Values	Algorithm	Parameters	Values
iTrees	50, 100, 200	n-gram	n	3, 5, 10, 20
Contamination	1-5%, 1-7%			

Table 6.9: Anomaly detection and feature extraction parameters for the second iteration.

6.5.2 Iteration 2: Evaluation

Table 6.10 includes all the new results generated in this iteration. The results strongly indicate that out of the four evaluated n -values, $n = 3$ and $n = 5$ outperform the larger values. Thus, larger values of n do not seem beneficial in the scenarios evaluated in this thesis and will consequently be excluded in further iterations.

Cont	iTrees	n=3		n=5		n=10		n=20	
		Detected	FPR	Detected	FPR	Detected	FPR	Detected	FPR
1-5%	50	90.91%	1.73%	81.82%	1.48%	86.36%	1.82%	77.27%	1.75%
	100	86.36%	1.81%	86.36%	1.58%	86.36%	1.75%	68.18%	1.85%
	200	81.82%	1.75%	86.36%	1.56%	77.27%	1.74%	68.18%	1.81%
1-7%	50	90.91%	2.16%	81.82%	1.89%	86.36%	2.14%	77.27%	2.11%
	100	86.36%	2.11%	86.36%	1.90%	86.36%	2.10%	68.18%	2.14%
	200	81.82%	2.09%	86.36%	1.89%	77.27%	2.09%	68.18%	2.18%
Average		86.36%	1.94%	84.85%	1.72%	83.33%	1.94%	71.21%	1.97%

Table 6.10: Evaluation results second iteration: n -gram feature extraction.

6.6 Iteration 3

Based on the previous iterations, the contamination mode and its range appear to have a meaningful impact on both the number of attacks detected and the FPR. This iteration focuses on evaluating different contamination ranges as input for the *custom* contamination mode in order to identify the best performing range(s).

6.6.1 Iteration 3: Parameters

The combination of feature extraction parameters used in this iteration comprises the top three performing sliding window combinations and the two best performing n-gram sizes from previous iterations. A fourth sliding window parameter combination ($W = 11$, $L = 5$) is added to evaluate the proposed window and step size relation of ($W = 2l + 1$, $L = l$), as proposed by Lee and Stolfo [37]. Six feature extraction parameters, fifteen contamination ranges and three counts of iTrees are evaluated in this iteration.

A total of 270 combinations is evaluated in this iteration. All parameter values evaluated in this iteration are listed in table 6.11.

iForest		Feature Extraction		
Parameters	Values	Algorithm	Parameters	Values
iTrees	50, 100, 200	n-gram	n	3, 5
Contamination	1%, 1-2%, 1-3%, 1-4%, 1-5%, 2%, 2-3%, 2-4%, 2-5%, 3%, 3-4%, 3-5%, 4%, 4-5%, 5%	Sliding Window	(W, L)	(10,5), (11,5), (11,6), (11,11)

Table 6.11: Anomaly detection and feature extraction parameters for the third iteration.

6.6.2 Iteration 3: Evaluation

The results detailed in table 6.12 includes the total top ten of all the evaluations performed for each of the feature extraction algorithms included. All ten sliding window combinations listed manage to detect all 22 attacks with an FPR roughly ranging from 3% to 4%. The evaluated n-gram combinations did not manage to detect all attacks using the same parameter values. While it provides a lower FPR, it does not detect all 22 attacks combined with any contamination input evaluated as part of this iteration. In a context where finding 21 out of 22 attacks is considered satisfactory, the three best performing parameter value combinations from each feature extraction algorithm are listed in table 6.13.

In this iteration, a total of 159 combinations managed to detect 21 attacks with an FPR ranging between 0.93% and 4.88%. A deeper analysis of the results showed that they all fail to detect the same attack, a denial of service attack causing NTP (CVE-2016-7434) to crash instantaneously. These results indicate that the sliding

window outperforms the n-grams for all the scenarios evaluated. As a result, the n-gram feature extraction is omitted for the following iterations.

Sliding Window						N-gram				
Cont	iTrees	Window Size	Step Size	Detected	FPR	Cont	iTrees	N-gram size	Detected	FPR
3%	50	11	6	100.00%	2.85%	2%	100	5	95.45%	1.71%
3%	200	10	5	100.00%	2.89%	2%	50	5	95.45%	1.85%
3-4%	50	11	6	100.00%	3.02%	2-3%	100	5	95.45%	1.91%
3-4%	200	10	5	100.00%	3.07%	2-3%	50	5	95.45%	2.02%
3-5%	50	11	6	100.00%	3.21%	2-4%	100	5	95.45%	2.14%
3-5%	200	10	5	100.00%	3.23%	2-4%	50	5	95.45%	2.21%
4%	50	10	5	100.00%	3.84%	2-5%	50	5	95.45%	2.35%
4%	200	10	5	100.00%	3.85%	2-3%	200	3	95.45%	2.36%
4%	50	11	5	100.00%	3.87%	2-5%	100	5	95.45%	2.36%
4%	50	50	26	100.00%	3.87%	3%	100	3	95.45%	2.40%

Table 6.12: Evaluation results third iteration: n-grams and sliding window feature extraction.

Sliding Window						N-gram				
Cont	iTrees	Window Size	Step Size	Detected	FPR	Cont	iTrees	N-gram size	Detected	FPR
1%	200	11	5	95.45%	0.93%	2%	100	5	95.45%	1.71%
1%	50	11	5	95.45%	0.93%	2%	50	5	95.45%	1.85%
1%	50	11	6	95.45%	0.94%	2-3%	100	5	95.45%	1.91%

Table 6.13: Evaluation results third iteration: The three best performing (FPR-wise) combinations for each feature extraction algorithm managing to detect 21 out of 22 attacks.

6.7 Iteration 4

The fourth and final iteration focuses on the number of iTrees and their impact on performance. Zou et al. [82] conducted a similar evaluation, where they concluded no improved detection performance was attained by increasing the number of iTrees beyond 100. This iteration is intended to investigate if the same conclusion can be drawn when using system calls as the data source.

6.7.1 Iteration 4: Parameters

This iteration includes the four best performing sliding window parameter values and contamination ranges from previous iterations. All possible combinations are evaluated with three new values of iTrees; 150, 250 and 500. A total of 36 new combinations is evaluated in this iteration, and the parameter combinations are listed in table 6.14.

6.7.2 Iteration 4: Evaluation

Table 6.15 lists the top ten performing combinations of this iteration. The top combination manages to detect all attacks with a false positive rate almost equal

iForest		Feature Extraction		
Parameters	Values	Algorithm	Parameters	Values
iTrees	150, 250, 500	Sliding Window	(W, L)	(10,5), (11,5), (11,6), (11,11)
Contamination	3%, 3-4%, 3-5%, 4%			

Table 6.14: Anomaly detection and feature extraction parameters for the fourth iteration.

to the top performer of the previous iteration (+0.02%). The results show that a larger number of estimators does not improve detection performance, confirming the findings of Zou et al. [82].

Sliding Window					
Cont	iTrees	Window Size	Step Size	Detected	FPR
3%	150	10	5	100.00%	2.87%
3-4%	150	10	5	100.00%	3.05%
3-5%	150	10	5	100.00%	3.18%
4%	500	10	5	100.00%	3.83%
4%	150	10	5	100.00%	3.83%
4%	250	10	5	100.00%	3.84%
4%	500	11	5	100.00%	3.84%
4%	150	11	6	100.00%	3.85%
3%	250	11	5	95.45%	2.85%
3%	150	11	6	95.45%	2.86%

Table 6.15: Evaluation results for the fourth iteration using isolation forest with sliding window feature extraction.

6.8 Refined Monitoring Technique

The refined monitoring technique was selected as the top performing technique of all techniques evaluated, based primarily on detection rate and secondarily on FPR. The refined technique is composed of **system calls** as the data source, feature extraction using a **sliding window** and anomaly detection using **iForest**. The parameter values of the refined technique are listed in table 6.16.

Component	Parameter	Optimal Value
iForest	iTrees	50
	Contamination	3%
Sliding Window	Window Size	11
	Step Size	6

Table 6.16: Parameters of the top performing technique.

The refined technique is capable of detecting all 22 attacks with an average FPR of 2.85%, clearly outperforming the baseline technique which manages to detect

18 attacks with an average FPR of 7.37%. From a STRIDE perspective, both techniques are capable of detecting the same four categories: TIDE. A detailed comparison of the techniques is presented in table 6.17. The evaluation results of all evaluated techniques can be found in appendix A.

Threat Impact	CVE ID	Refined		Baseline		STRIDE
		Detected	FPR	Detected	FPR	
Return Shell and Execute Arbitrary Code	CVE-2014-3120	✓	2.90%	✗	1.41%	IE
	CVE-2015-3306	✓	2.99%	✗	2.45%	IE
	CVE-2017-7494	✓	3.00%	✗	0.09%	TE
	CVE-2016-3714	✓	2.95%	✓	1.34%	TIE
	CVE-2015-8562	✓	3.00%	✓	0.24%	IE
Execute Arbitrary Code	CVE-2017-12615	✓	2.99%	✓	9.94%	T
	CVE-2014-6271	✓	2.98%	✓	9.91%	IE
	CVE-2015-1427	✓	2.80%	✓	9.80%	IE
	CVE-2018-16509	✓	2.90%	✓	9.99%	IE
	CVE-2018-19475	✓	2.93%	✓	9.98%	IE
	CVE-2019-6116	✓	2.93%	✓	9.97%	IE
	CVE-2016-10033	✓	2.82%	✗	10.03%	T
	CVE-2017-5638	✓	2.98%	✓	9.40%	I
Escalation of Privilege	CVE-2017-12635	✓	3.01%	✓	0.92%	E
Disclose Credential Information	CVE-2015-5531	✓	2.93%	✓	10.20%	I
	CVE-2016-1897	✓	2.94%	✓	7.99%	I
	CVE-2017-8917	✓	2.59%	✓	9.98%	I
	CVE-2017-7529	✓	1.79%	✓	16.02%	I
	CVE-2017-5223	✓	3.00%	✓	9.48%	I
Crash the Application	CVE-2015-5477	✓	2.79%	✓	10.00%	D
	CVE-2016-7434	✓	2.48%	✓	3.10%	D
Consume Excessive CPU	CVE-2014-0050	✓	2.98%	✓	9.87%	D
Average		100.00%	2.85%	81.82%	7.37%	

Table 6.17: Detailed comparison of baseline and refined monitoring technique.

7

Discussion

This chapter first discuss how containers relate to Firecracker, and why the results of this thesis are of relevance. Subsequent sections discuss how the thesis relates to and answers the research questions declared in section 1.2.

7.1 Containers vs Firecracker

Amazon has made recent efforts to develop a virtualisation technology that promises to provide the benefits of both containers and VMs, without any of their drawbacks, named Firecracker (described in section 2.1.3). Firecracker aims to use heavily optimised VMs (MicroVMs) to provide a virtualisation solution with the performance and flexibility of containers and the isolation of VMs. Given the features promised by Firecracker, the relevancy of conducting further research in the field of containers could be questioned.

Arguably, improving the security of containers should still be considered important for three reasons. First, allowing practitioners a choice should always be considered a benefit. Second, Firecracker was recently released and still needs time to prove itself as there is a risk of unforeseen issues or security-related flaws. Third, containers have seen widespread adoption since the release of Docker, and the technology now has a firm grip on the industry. Even if Firecracker was to become the defacto standard in the future, it is likely to take time. In the meantime, improving the security of containers should still be considered an essential step towards increasing the overall security of virtualised systems.

7.2 Existing Monitoring Techniques

RQ1. *What current monitoring techniques exist that might be applicable for the container-based virtualisation environment?*

The first research question asks for the currently available monitoring techniques that might apply to monitoring containers. The approach to answering this question was in the form of a literature review. The literature review yielded a summary of existing monitoring techniques applicable to containers. The identified techniques

and their evaluation results are summarised and presented in section 4.1.1. These techniques and evaluations guided the effort towards developing and evaluating the refined monitoring technique, which could be considered as a combination of multiple techniques found during the review.

The citation matrix, as seen in table 4.7, generated from the final set of included studies, is arguably a bit sparse. We attribute this to three influential factors; (I) Strict selection criteria; the requirement of each paper explicitly evaluating their techniques using containers filtered out a large portion of the references; (II) Relatively new field of research; the earliest publication found fitting the specified selection criteria was published in 2015. A significant portion of the included papers had been, at the time, cited less than ten times according to Google Scholar; and (III) The decision to include the most comprehensive version of a study resulted in multiple exclusions, most notably of studies authored by Areeg and Pahl. A total of eight studies [57, 55, 58, 56, 59, 60, 61, 62] and one dissertation [7] written by these authors was found during the snowballing procedure. The dissertation included the combined efforts of the seven previously published papers, making it the most comprehensive version. As a result, the dissertation was included, and the seven previously published papers were excluded. Many of the included studies referenced at least one of the eight excluded studies, but not the dissertation, reducing the density of the citation matrix.

An improved selection criterion could be to include the paper with the most citations instead of the most comprehensive version, which would likely result in a denser matrix. However, since the primary purpose of the literature review was to investigate what techniques are currently available to serve as a basis for improvement, this alternative is arguably less suitable. To evaluate the validity of the literature review, an inter-rater reliability test was performed, presented in section 4.2. Said test yielded *almost perfect agreement*, indicating well-defined selection criteria with sufficient consistency.

7.3 Improving a Technique for Container Security Monitoring

RQ2. *What are the steps needed to be taken in order to develop an improved monitoring technique with additional layers of security and increased effectiveness?*

The second research question focuses on the path towards developing an improved monitoring technique. The first step is to formulate a planned approach followed by understanding the current state of the research field and available techniques. In the context of this research, this step was accomplished by conducting a literature review, followed by data extraction and synthesis to provide a condensed overview. The next step involves creating a simulation environment and data collection tool to generate and record behavioural data for evaluation. Heimdall was developed for said data collection, allowing effortless logging and labelling of a containerised

application's system calls. The following step involves the design of the technique and choosing feature extraction algorithms. As part of this research, an effort was made to evaluate as many feature extraction algorithms as the limited amount of time would allow.

The selection of technique compositions to evaluate was based on an analysis of the information gathered from the literature review. The compositions were then iteratively evaluated and tuned by changing parameters and hyper-parameters of their respective algorithms. To facilitate a streamlined evaluation process, the framework Hlin was developed. Hlin assists the evaluation of multiple techniques and exports the results for further analysis.

The process summarised in this section generalises the process and approach used as part of this thesis, i.e. the steps taken by the authors of this thesis to provide a technique with improved efficiency over the stated baseline. The process outlined is by no means to be considered the only nor the best approach.

Using the design science research methodology to refine a monitoring technique proved itself to be incredibly useful. The provision of a method that describes *how* to approach a problem in order to improve or create a solution by first understanding the problem domain and then iteratively design and evaluate solutions helped tremendously. In the context of this thesis, DSRM guided the inclusion of a literature review in order to explore the problem space and the iterative refinement process. Compared to the studies included in the literature review, which tend only to state the problem, solution and evaluation of their techniques, using DSRM provides better insight and understanding of the problem and solution in question. Given that the authors of the included studies most likely have adequate knowledge and experience in this field, they might consider DSRM more of a hindrance than something useful. Based on this reasoning, speculation is made that DSRM is better suited for problems where the problem domain is large or unknown to the researchers or where the solution space has not been scoped to only include specific, pre-determined solutions.

However, the solution space in this thesis was partly scoped as well. As decisions were made only to implement specific techniques and then iteratively evaluate parameters, whereas solutions based on other techniques were excluded. For instance, monitoring resource utilisation and using rule-based intrusion detection were excluded early. The scoping was deemed necessary because of time constraints; however, exploring other solutions could further improve container monitoring and anomaly detection.

The refined technique was developed by making design-decisions based on data analysis from either evaluations or the literature review. Based on the literature review data, decisions were made during the early stages of solution design to establish a foundation that was to be iteratively refined. Because of the reduced scope of the literature review, no quantitative analysis of the included evaluations to statistically identify the most promising candidates was conducted. Instead, they were identified

by reasoning about the evaluations, which can provide better context but also introduces bias. Thus, a potential improvement to the foundation establishment would be quantitative data analysis in the literature review, which would provide additional objectivity to the decision-making. The same can be argued for the decisions made during the iterative refinement process, which were primarily based on evaluation data from previous iterations. The selection of parameter value combinations to evaluate for the subsequent iteration was based on reasoning, meaning there is no guarantee the *correct* selections were made.

The refined monitoring technique presented in this thesis was developed iteratively, using the knowledge attained by conducting a literature review as a foundation. The technique monitors the system calls invoked from a container, performs feature extraction using the sliding window algorithm, and encodes it to a Bag of System Calls (BoSC) representation. The detection of anomalous behaviour is then achieved using the Isolation Forest algorithm to train a model used to perform predictions on sequences of BoSC.

7.4 Evaluation of the Proposed Technique

RQ3. *How well does the refined monitoring technique perform with respect to STRIDE model coverage and performance, e.g. ability to detect threats?*

The fundamental aim of this thesis is to design, implement and evaluate a non-intrusive application-agnostic container monitoring technique. In order to accomplish this, an effort was made to include a diverse set of containerised applications. The inclusion of STRIDE assisted the selection of threats included in the simulation. With the intent of improving the significance of the evaluations, exploits were carefully selected to ensure the creation of a diverse simulation environment in terms of STRIDE. Attacks that could be categorised into multiple STRIDE categories, such as remote code execution, were tailored to fall into a particular category.

However, the **Spoofing** and **Repudiation** categories of STRIDE were not included in the simulation due to the nature of the threats and the source of data. These types of threats involve performing common actions posing as another user. Since the collected system calls do not contain any information that can be used to distinguish between actors, it is next to impossible to detect these types of threats.

Since both the baseline and the refined technique detect at least one attack of each included STRIDE category, the refined technique does not perform better from a strict STRIDE point of view. However, to state that any monitoring technique covers all categories of STRIDE in a complete sense would be false since that is not possible to prove. Arguably, using STRIDE as a metric to evaluate monitoring techniques is a poor choice, if not even counter-productive. STRIDE is not to blame here, though, as the model was developed to help reasoning about a system and the potential threats it is exposed to [64].

The refined technique provides improved performance by detecting all 22 attacks compared to the baseline detecting only 18 (compared in section 6.8). In addition to the superior detection performance, the average FPR of the refined technique amounts to 2.85%, compared to the baseline's 7.37%. These results were achieved iteratively as explained in chapter 6. Each iteration aimed to examine the performance impact of different parameters and their values, with the intent of excluding some parameter, parameter value or feature extraction algorithm in each iteration. The results presented in section 6.8 far exceeds the performance of the baseline and are considered an improvement. However, the presented result should not be viewed as the best possible result attainable using the algorithms and parameters evaluated. There is likely a better set of parameter values that exist for the presented technique, as the evaluated sets are restricted to a relatively small number of parameter values, e.g. only six sliding window parameter combinations were evaluated. Conceivably, another parameter value could perform even better. However, the potential improvements to be found by further extending the parameter value space would suffer from diminishing returns and would likely not yield a justifiable return on time invested.

The best performing combination of values included a fixed contamination value of 3%. Using a fixed contamination rate is comparable to the p -parameter used in the baseline technique's anomaly detection algorithm, as both effectively act as thresholds. A higher value is more likely to detect more anomalies at the cost of a higher FPR. Thus, in a setting where it is of higher interest to keep a low rate of false alarms, other parameter values should be evaluated. Similarly, using a different set of containerised applications, the results would be expected to vary to some extent. However, given the diversity of the containerised applications, it would be reasonable to expect the proposed technique to offer better performance compared to the baseline.

8

Threats to Validity

This chapter describes the identified threats to validity and how they were addressed or mitigated.

8.1 Construct Validity

The data used for both training and evaluation is collected, transformed and evaluated using software developed specifically for this research. During any software development, the presence of unknown bugs is always a potential risk. These bugs could alter the data by, for example, not logging *every* system calls. To mitigate the risk of generating incorrect data, the developed framework uses several libraries and tools which provide the desired functionality. The open-source system visibility tool Sysdig handles the interception and extraction of system calls. Sysdig has been used successfully in numerous other studies utilising system call traces for monitoring system behaviour [41, 5, 17, 70], adding validity to the use of the tool. The scikit-learn library handles the implementation of the kNN and Isolation Forest algorithm and the confusion-matrix calculations. Using a reputable and widely adopted open-source library [63] instead of developing a new solution with the same purpose minimises the risk of invalid results caused by software bugs.

8.2 Internal Validity

The use of Google Scholar does mitigate the risk of introducing publisher bias [79]. However, some inconsistencies were observed during the creation of the starting set for the literature review. When using identical search strings, the results shown to the authors differed slightly. Generally, the same results were shown on the same page only in a different order. However, some cases were observed where a paper was listed on a different page, threatening the reproducibility of the resulting list of included papers. This inconsistency of the served results means that the search algorithm is an unknown factor capable of affecting the outcome.

Another threat is the potential of an unknown factor interfering with the data captured. The following strategies are used to mitigate these kinds of threats; (I) To mitigate any impact made on the performance of the system during the generation

of data, only the necessary applications were running during the generation and capturing of data; (II) All evaluations were performed on the same machine; and (III) Using the open-source system visibility tool Sysdig for capturing the system calls.

A third threat to the internal validity is values of the parameters and hyper-parameters used to achieve the results. To evaluate every possible value for every parameter is arguably not feasible nor likely to be worth the effort. Nevertheless, there is a likelihood that other values not evaluated would outperform the values presented as *best performing* as part of this thesis.

8.3 External Validity

The literature review conducted in this study, detailed in section 3.1, followed the guidelines provided by Wohlin [79]. Wohlin states that this approach is not necessarily intended to replace other methods of literature review but rather extend them. As the snowballing search served as our primary approach for identifying techniques and studies applicable to our research, this study does not strictly confer with said guidelines. The decision was a conscious one, with the intent of keeping the scope of the review reasonable. However, it does introduce the risk of not providing a complete picture of the current state and all available techniques of interest. As mentioned in section 3.1.2, using Google Scholar during the generation of the starting set does help mitigate publisher bias and consequently provides a more comprehensive picture than if using a specific publishers' database. However, even if publisher bias is mitigated entirely, the impact of the starting set could still be considered unbalanced since a differing starting set could potentially result in a relatively different final set. The sparse citation matrix further highlights this. This imbalanced impact does stem from the snowballing procedure itself and cannot be avoided unless paired with other methods.

The data used for evaluation is another important threat. Since the normal data is generated by a synthetic load rather than real-world data generated by actual users, the system call traces used in this thesis will likely differ somewhat from traces generated in a real-world scenario. With the intent of mitigating this issue to a reasonable extent, an effort was made to ensure the synthetic load includes the most common actions a real user would want to perform with each application. To further mitigate the issue, the load was varied during data collection, alternating between different levels of load to reflect a fluctuating number of simultaneous users.

Another threat to external validity is that the performance of the refined technique heavily depends on the type of threat or application running within the monitored container. The strategy employed to mitigate this threat was to include as many containerised applications as the time constraint allowed for while making an effort to ensure a diverse set of containerised applications. Still, there is a possibility that the performance would not be replicated using other applications and threats.

The choice of refining a machine learning-based approach in itself is another mitigation strategy employed to enhance the generalisability of the results. Compared to a solution employing a signature-based, static detection of threats and vulnerabilities, the performance of the solution presented is not dependant on the threats or vulnerabilities signatures being known or previously detected.

9

Conclusion

Container-based virtualisation is an increasingly popular virtualisation technique, mainly due to the flexibility and efficiency it brings when compared to the more traditional VM-based approach. However, containerisation introduces major security-related risks in the form of weakened isolation, which can be partly mitigated through solid monitoring and anomaly detection. The aim of this thesis was to examine the current landscape of container security monitoring and investigate the process of refining a monitoring technique, including the implementation and evaluation of a novel technique. The contributions of this thesis are the following:

- **An overview of available state-of-the-art container monitoring techniques:** By conducting a literature review using the snowballing search approach, currently available techniques have been synthesised and presented to facilitate comparison and assist future technique selection.
- **A method of refining a container monitoring technique:** An iterative process of refining a technique using the DSRM is described and presented, including designing, implementing and evaluating techniques. The process can be imitated by future research to refine techniques.
- **Heimdall - A container system call monitoring tool:** Heimdall facilitates effortless collection and labelling of system calls invoked by containers for anomaly detection using machine learning. Both practitioner and researchers can utilise the tool to collect system call logs from containers.
- **Hlin - An anomaly detector evaluation framework:** Hlin expedites the process of implementing, training and evaluating anomaly detection techniques. The iterative refinement process is streamlined by providing a simple evaluation of different technique compositions and parameter combinations. Researchers can employ the framework to perform an offline evaluation of anomaly detection techniques.
- **A well-performing refined container monitoring technique:** The refined technique is novel, unsupervised, non-intrusive and application-agnostic. The technique uses system calls as the data source, a sliding window for feature extraction and the iForest algorithm for anomaly detection, a composition previously unseen. It provides a 100% detection rate and a relatively low FPR of 2.85%, a substantial improvement over the evaluated baseline technique.

This research benefits researchers by providing additional knowledge and evaluation

of a technique not previously evaluated for monitoring containers. This information could inform decisions of what algorithms and compositions to investigate further or disregard in future research.

This thesis benefits both practitioners and researchers by providing an anomaly detector evaluation framework and a container system call monitoring tool. These could be used to evaluate other algorithms and technique compositions for use with containers. Additionally, practitioners could evaluate the performance of the refined technique presented with applications of their choice, potentially enhancing the security of their deployed containerised applications.

9.1 Future Work

This thesis provides a baseline for a continuous investigation of how to best monitor containers to improve their security. The evaluation of this thesis is based on the detection performance using synthetically generated data. Confirming the performance of the refined technique in a real-world deployment would further validate the proposed monitoring technique.

The evaluation presented is based on pre-generated system call traces. Future researchers could investigate how the performance is affected by real-time detection. Theoretically, the technique should yield performance similar to the results presented as part of this thesis. However, the lack of real-time evaluation calls for future investigation.

Other possible areas for future research would be to evaluate the performance using a different set of containerised applications. While an effort was made to include a diverse set of applications, a more extensive set could provide further insights.

Bibliography

- [1] Amr S. Abed, Charles Clancy, and David S. Levy. “Intrusion Detection System for Applications using Linux Containers”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9331 (Nov. 2016), pp. 123–135. ISSN: 16113349. DOI: 10.1007/978-3-319-24858-5_8. arXiv: 1611.03056. URL: <http://arxiv.org/abs/1611.03056>.
- [2] Amr S. Abed, T. Charles Clancy, and David S. Levy. “Applying Bag of System Calls for Anomalous Behavior Detection of Applications in Linux Containers”. In: *2015 IEEE Globecom Workshops (GC Wkshps)*. IEEE, Dec. 2015, pp. 1–5. ISBN: 978-1-4673-9526-7. DOI: 10.1109/GLOCOMW.2015.7414047. arXiv: 1611.03053. URL: <http://ieeexplore.ieee.org/document/7414047/>.
- [3] Alexandru Agache, Marc Brooker, Andreea Florescu, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, Implementation Nsdi, Alexandra Iordache, Marc Brooker, Anthony Liguori, Rolf Neugebauer, and Phil Piwonka. “Firecracker : Lightweight Virtualization for Serverless Applications”. In: *Nsdi '20* (2020), pp. 419–434. URL: <https://www.usenix.org/conference/nsdi20/presentation/agache>.
- [4] Suaad S. Alarifi and Stephen D. Wolthusen. “Detecting anomalies in IaaS environments through virtual machine host system call analysis”. In: *2012 International Conference for Internet Technology and Secured Transactions, ICITST 2012*. 2012, pp. 211–218. ISBN: 9781908320087.
- [5] Mohammed J Aljebreen. “Towards Intelligent Intrusion Detection Systems for Cloud Computing”. PhD thesis. 2018. URL: <https://repository.lib.fit.edu/handle/11141/2554>.
- [6] AMD. *Virtualization solutions*. URL: <https://www.amd.com/en/technologies/virtualization-solutions> (visited on 02/12/2021).
- [7] Samir Areeg and Pahl Claus. “Monitoring, Detecting, Identifying, and Healing Anomalous Workload in Clustered Computing Environments”. PhD thesis. Apr. 2020. DOI: 10.13140/RG.2.2.20110.59206.
- [8] AWS. *Linux AMI virtualization types - Amazon Elastic Compute Cloud*. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/virtualization_types.html (visited on 06/17/2021).
- [9] Anthony Bettini. *Vulnerability Exploitation in Docker Container Environments*. 2015. URL: <https://www.slideshare.net/FlawCheck/vulnerability-exploitation-in-docker-container-environments>.

- [10] Kelly Brady, Seung Moon, Tuan Nguyen, and Joel Coffman. “Docker Container Security in Cloud Computing”. In: *2020 10th Annual Computing and Communication Workshop and Conference, CCWC 2020*. Institute of Electrical and Electronics Engineers Inc., Jan. 2020, pp. 975–980. ISBN: 9781728137834. DOI: 10.1109/CCWC47524.2020.9031195.
- [11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. “Anomaly Detection: A Survey”. In: *ACM Comput. Surv.* 41 (2009). DOI: 10.1145/1541880.1541882.
- [12] Gary Chen. *Modernizing Applications with Containers in the Public Cloud*. 2019. URL: <https://www.ibm.com/downloads/cas/AWPB0Y7L> (visited on 03/18/2021).
- [13] Sung Bae Cho and Hyuk Jang Park. “Efficient anomaly detection by modeling privilege flows using hidden Markov model”. In: *Computers and Security* 22.1 (Jan. 2003), pp. 45–55. ISSN: 01674048. DOI: 10.1016/S0167-4048(03)00112-3.
- [14] Z. Cliffe Schreuders, Tanya McGill, and Christian Payne. “Empowering end users to confine their own applications: The results of a usability study comparing SELinux, AppArmor, and FBAC-LSM”. In: *ACM Transactions on Information and System Security* 14.2 (Sept. 2011). ISSN: 10949224. DOI: 10.1145/2019599.2019604. URL: <https://dl.acm.org/doi/abs/10.1145/2019599.2019604>.
- [15] William W. Cohen. “Fast Effective Rule Induction”. In: *Machine Learning Proceedings 1995*. Elsevier, Jan. 1995, pp. 115–123. DOI: 10.1016/b978-1-55860-377-6.50023-2.
- [16] Theo Combe, Antony Martin, and Roberto Di Pietro. “To Docker or Not to Docker: A Security Perspective”. In: *IEEE Cloud Computing* 3.5 (Sept. 2016), pp. 54–62. ISSN: 2325-6095. DOI: 10.1109/MCC.2016.100.
- [17] Pinchen Cui. *DevSecOps of Containerization*. Tech. rep. Aug. 2020. URL: <https://etd.auburn.edu/handle/10415/7425>.
- [18] Docker. *Docker overview / Docker Documentation*. 2018. URL: <https://docs.docker.com/get-started/overview/> (visited on 04/07/2021).
- [19] Qingfeng Du, Tiandi Xie, and Yu He. “Anomaly Detection and Diagnosis for Container-Based Microservices with Performance Monitoring”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 11337 LNCS. Springer Verlag, 2018, pp. 560–572. ISBN: 9783030050627. DOI: 10.1007/978-3-030-05063-4_42.
- [20] Emelie Engström, Margaret Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. “How software engineering research aligns with design science: a review”. In: *Empirical Software Engineering* 25.4 (July 2020), pp. 2630–2660. ISSN: 15737616. DOI: 10.1007/s10664-020-09818-7. arXiv: 1904.12742.
- [21] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. “An updated performance comparison of virtual machines and Linux containers”. In: *ISPASS 2015 - IEEE International Symposium on Performance Analysis of Systems and Software*. Institute of Electrical and Electronics Engineers Inc.,

- Apr. 2015, pp. 171–172. ISBN: 9781479919567. DOI: 10.1109/ISPASS.2015.7095802.
- [22] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. “Sense of self for unix processes”. In: *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*. 1996, pp. 120–128. DOI: 10.1109/secpri.1996.502675.
 - [23] Mohamed Hedi Fourati, Soumaya Marzouk, Khalil Drira, and Mohamed Jmaiel. “DOCKERANALYZER : Towards Fine Grained Resource Elasticity for Microservices-Based Applications Deployed with Docker”. In: *2019 20th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*. IEEE, Dec. 2019, pp. 220–225. ISBN: 978-1-7281-2616-6. DOI: 10.1109/PDCAT46702.2019.00049. URL: <https://ieeexplore.ieee.org/document/9029138/>.
 - [24] Holger Gantikow, Christoph Reich, Martin Knahl, and Nathan Clarke. “Rule-Based Security Monitoring of Containerized Environments”. In: *Communications in Computer and Information Science*. Vol. 1218 CCIS. Springer, May 2020, pp. 66–86. ISBN: 9783030494315. DOI: 10.1007/978-3-030-49432-2_4.
 - [25] Li Zhong Geng and Hui Bo Jia. “A low-cost method to intrusion detection system using sequences of system calls”. In: *2009 2nd International Conference on Information and Computing Science, ICIC 2009*. Vol. 1. 2009, pp. 143–146. ISBN: 9780769536347. DOI: 10.1109/ICIC.2009.43.
 - [26] Xuan Dau Hoang, Jiankun Hu, and Peter Bertok. “A multi-layer model for anomaly intrusion detection using program sequences of system calls”. In: *IEEE International Conference on Networks, ICON*. 2003, pp. 531–536. ISBN: 0780377885. DOI: 10.1109/icon.2003.1266245. URL: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.154.6718>.
 - [27] Steven A. Hofmeyr, Stephanie Forrest, and Anil Somayaji. “Intrusion detection using sequences of system calls”. In: *Journal of Computer Security* 6.3 (1998), pp. 151–180. ISSN: 0926227X. DOI: 10.3233/JCS-980109.
 - [28] Intel. *Intel® Virtualization Technology*. 2018. URL: <https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html>.
 - [29] Dae Ki Kang, Doug Fuller, and Vasant Honavar. “Learning classifiers for misuse and anomaly detection using a bag of system calls representation”. In: *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005*. Vol. 2005. 2005, pp. 118–125. ISBN: 0780392906. DOI: 10.1109/IAW.2005.1495942. URL: <https://ieeexplore.ieee.org/abstract/document/1495942>.
 - [30] Rupesh Raj Karn, Prabhakar Kudva, Hai Huang, Sahil Suneja, and Ibrahim M. Elfadel. “Cryptomining Detection in Container Clouds Using System Calls and Explainable Machine Learning”. In: *IEEE Transactions on Parallel and Distributed Systems* 32.3 (Mar. 2021), pp. 674–691. ISSN: 1045-9219. DOI: 10.1109/TPDS.2020.3029088. URL: <https://ieeexplore.ieee.org/document/9215018/>.

- [31] Eric Knauss. *Constructive master's thesis work in industry: guidelines for applying design science research*. Dec. 2020. arXiv: 2012.04966. URL: <http://arxiv.org/abs/2012.04966>.
- [32] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. "Spectre attacks: Exploiting speculative execution". In: *Proceedings - IEEE Symposium on Security and Privacy*. Vol. 2019-May. Institute of Electrical and Electronics Engineers Inc., May 2019, pp. 1–19. ISBN: 9781538666609. DOI: 10.1109/SP.2019.00002.
- [33] Loren Kohnfelder and Praerit Garg. "The threats to our products". In: *Microsoft Security Development Blog* (1999), p. 3. URL: <https://www.microsoft.com/security/blog/2009/08/27/the-threats-to-our-products/>.
- [34] J. Richard Landis and Gary G. Koch. "The Measurement of Observer Agreement for Categorical Data". In: *Biometrics* 33.1 (Mar. 1977), p. 159. ISSN: 0006341X. DOI: 10.2307/2529310.
- [35] Steve G. Langer and Todd French. "Virtual machine performance benchmarking". In: *Journal of Digital Imaging* 24.5 (Oct. 2011), pp. 883–889. ISSN: 08971889. DOI: 10.1007/s10278-010-9358-6.
- [36] Pavel Laskov, Patrick Dussel, Christin Schafer, and Konrad Rieck. "Learning intrusion detection: Supervised or unsupervised?" In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3617 LNCS. September 2005 (2014), pp. 50–57. ISSN: 03029743. DOI: 10.1007/11553595_6.
- [37] Wenke Lee and Salvatore J Stolfo. "Data Mining Approaches for Intrusion Detection". In: *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*. SSYM'98. USA: USENIX Association, 1998, p. 6.
- [38] Wei Li. "Using genetic algorithm for network intrusion detection". In: (2004).
- [39] Zhengmin Li, Zhaoxin Zhang, Xinran Liu, and Chungue Zhu. "Anomaly Detection for Container Cluster based on JointCloud Platform". In: *Proceedings of the 2019 3rd International Conference on Compute and Data Analysis*. New York, NY, USA: ACM, Mar. 2019, pp. 26–30. ISBN: 9781450366342. DOI: 10.1145/3314545.3314567. URL: <https://doi.org/10.1145/3314545.3314567>.
- [40] Hung Jen Liao, Chun Hung Richard Lin, Ying Chih Lin, and Kuang Yuan Tung. *Intrusion detection system: A comprehensive review*. Jan. 2013. DOI: 10.1016/j.jnca.2012.09.004.
- [41] Yuhang Lin, Olufogorehan Tunde-Onadele, and Xiaohui Gu. "CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications". In: *ACM International Conference Proceeding Series*. Vol. 10. 2020. Association for Computing Machinery, Dec. 2020, pp. 179–188. ISBN: 9781450388580. DOI: 10.1145/3427228.3427236. URL: <https://doi.org/10.1145/3427228.3427236>.
- [42] Linux Contributors. *namespaces(7) - Linux manual page*. 2019. URL: <https://man7.org/linux/man-pages/man7/namespaces.7.html> (visited on 04/06/2021).

- [43] Linux Manual Page. *sysdig(8) - Linux manual page*. URL: <https://man7.org/linux/man-pages/man8/sysdig.8.html> (visited on 04/27/2021).
- [44] Fei Tony Liu, Kai Ming Ting, and Zhi Hua Zhou. “Isolation forest”. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*. 2008, pp. 413–422. ISBN: 9780769535029. DOI: 10.1109/ICDM.2008.17.
- [45] Zhen Liu, Nathalie Japkowicz, Ruoyu Wang, Yongming Cai, Deyu Tang, and Xianfa Cai. “A statistical pattern based feature extraction method on system call traces for anomaly detection”. In: *Information and Software Technology* 126 (Oct. 2020), p. 106348. ISSN: 09505849. DOI: 10.1016/j.infsof.2020.106348.
- [46] Chengzhi Lu, Kejiang Ye, Wenyan Chen, and Cheng-Zhong Xu. “ADGS: Anomaly Detection and Localization Based on Graph Similarity in Container-Based Clouds”. In: *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. Vol. 2019-Decem. IEEE, Dec. 2019, pp. 53–60. ISBN: 978-1-7281-2583-1. DOI: 10.1109/ICPADS47876.2019.00016. URL: <https://ieeexplore.ieee.org/document/8975844/>.
- [47] Mary L. McHugh. “Interrater reliability: The kappa statistic”. In: *Biochemia Medica* 22.3 (2012), pp. 276–282. ISSN: 13300962. DOI: 10.11613/bm.2012.031. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/>.
- [48] Mendeley. URL: <https://www.mendeley.com> (visited on 03/01/2021).
- [49] MEYER RA and SEAWRIGHT LH. “A virtual machine time-sharing system”. In: *IBM Systems Journal* 9.3 (1970), pp. 199–218. ISSN: 00188670. DOI: 10.1147/sj.93.0199.
- [50] NIST - National Institute of standards and technology. *NIST - National Vulnerability Database*. URL: <https://nvd.nist.gov/> (visited on 04/04/2021).
- [51] Animesh Patcha and Jung Min Park. “An overview of anomaly detection techniques: Existing solutions and latest technological trends”. In: *Computer Networks* 51.12 (Aug. 2007), pp. 3448–3470. ISSN: 13891286. DOI: 10.1016/j.comnet.2007.02.001.
- [52] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [53] David M. W. Powers. *Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation*. Tech. rep. December. 2007. URL: <https://arxiv.org/abs/2010.16061>.
- [54] Rajsimman Ravichandiran, Hadi Bannazadeh, and Alberto Leon-Garcia. “Anomaly Detection using Resource Behaviour Analysis for Autoscaling systems”. In: *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, June 2018, pp. 192–196. ISBN: 978-1-5386-4633-5. DOI: 10.1109/NETSOFT.2018.8460025. URL: <https://ieeexplore.ieee.org/document/8460025/>.
- [55] Areeg Samir, Nabil El Ioini, Ilenia Fronza, Hamid R Barzegar, Van Thanh Le, and Claus Pahl. “A Controller for Anomaly Detection, Analysis and Management for Self-Adaptive Container Clusters”. In: 2020.

- [56] Areeg Samir, Nabil El Ioini, Ilenia Fronza, Hamid R Barzegar, Van Thanh Le, and Claus Pahl. “Anomaly Detection and Analysis for Reliability Management in Clustered Container Architectures”. In: 2020.
- [57] Areeg Samir and Claus Pahl. “A Controller Architecture for Anomaly Detection, Root Cause Analysis and Self-Adaptation for Cluster Architectures”. In: 2019.
- [58] Areeg Samir and Claus Pahl. “Anomaly Detection and Analysis for Clustered Cloud Computing Reliability”. In: *The Eleventh International Conference on Cloud Computing, Grids, and Virtualization*. 5. May 2019, pp. 110–119.
- [59] Areeg Samir and Claus Pahl. “Detecting and localizing anomalies in container clusters using markov models”. In: *Electronics (Switzerland)* 9.1 (Jan. 2020). ISSN: 20799292. DOI: 10.3390/electronics9010064.
- [60] Areeg Samir and Claus Pahl. “Detecting and predicting anomalies for edge cluster environments using hidden markov models”. In: *2019 4th International Conference on Fog and Mobile Edge Computing, FMEC 2019*. Institute of Electrical and Electronics Engineers Inc., June 2019, pp. 21–28. ISBN: 9781728117966. DOI: 10.1109/FMEC.2019.8795337.
- [61] Areeg Samir and Claus Pahl. “DLA: Detecting and localizing anomalies in containerized microservice architectures using markov models”. In: *Proceedings - 2019 International Conference on Future Internet of Things and Cloud, FiCloud 2019*. Institute of Electrical and Electronics Engineers Inc., Aug. 2019, pp. 205–213. ISBN: 9781728128887. DOI: 10.1109/FiCloud.2019.00036.
- [62] Areeg Samir and Claus Pahl. “Self-adaptive healing for containerized cluster architectures with hidden Markov models”. In: *2019 4th International Conference on Fog and Mobile Edge Computing, FMEC 2019*. Institute of Electrical and Electronics Engineers Inc., June 2019, pp. 68–73. ISBN: 9781728117966. DOI: 10.1109/FMEC.2019.8795322.
- [63] Scikit-learn.org. *Who is using scikit-learn? — scikit-learn 0.24.2 documentation*. URL: <https://scikit-learn.org/stable/testimonials/testimonials.html> (visited on 05/12/2021).
- [64] Adam Shostack. *Threat modeling: Designing for security*. John Wiley & Sons, 2014.
- [65] Sachchidanand Singh and Nirmala Singh. “Containers & Docker: Emerging roles & future of Cloud technology”. In: *Proceedings of the 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology, iCATccT 2016*. Jan. 2017, pp. 804–807. ISBN: 9781509023981. DOI: 10.1109/ICATCCT.2016.7912109.
- [66] Vindeep Singh and Sateesh K. Peddoju. “Container-based microservice architecture for cloud applications”. In: *Proceeding - IEEE International Conference on Computing, Communication and Automation, ICCCA 2017*. Vol. 2017-Janua. Institute of Electrical and Electronics Engineers Inc., Dec. 2017, pp. 847–852. ISBN: 9781509064717. DOI: 10.1109/CCAA.2017.8229914.
- [67] Siddharth Srinivasan, Akshay Kumar, Manik Mahajan, Dinkar Sitaram, and Sanchika Gupta. “Probabilistic real-time intrusion detection system for docker containers”. In: *Communications in Computer and Information Sci-*

- ence. Vol. 969. Springer Verlag, Sept. 2019, pp. 336–347. ISBN: 9789811358258. DOI: 10.1007/978-981-13-5826-5_26.
- [68] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. “Container security: Issues, challenges, and the road ahead”. In: *IEEE Access* 7 (2019), pp. 52976–52996. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2911732.
- [69] Yongzhong Sun, Kejiang Ye, and Cheng-Zhong Xu. “PLMSys: A Cloud Monitoring System Based on Cluster Performance and Container Logs”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 12403 LNCS. Springer Science and Business Media Deutschland GmbH, 2020, pp. 111–125. ISBN: 9783030596347. DOI: 10.1007/978-3-030-59635-4_8.
- [70] Olufogorehan Tunde-Onadele, Jingzhu He, Ting Dai, and Xiaohui Gu. “A Study on Container Vulnerability Exploit Detection”. In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, June 2019, pp. 121–127. ISBN: 978-1-7281-0218-4. DOI: 10.1109/IC2E.2019.00026. URL: <https://ieeexplore.ieee.org/document/8790061/>.
- [71] VMware. *Hypervisor*. URL: <https://www.vmware.com/topics/glossary/content/hypervisor> (visited on 02/12/2021).
- [72] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. “Cost of virtual machine live migration in clouds: A performance evaluation”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 5931 LNCS. Springer, Berlin, Heidelberg, 2009, pp. 254–265. ISBN: 3642106641. DOI: 10.1007/978-3-642-10665-1_23.
- [73] Vulhub. *GitHub - vulhub/vulhub: Pre-Built Vulnerable Environments Based on Docker-Compose*. URL: <https://github.com/vulhub/vulhub> (visited on 04/26/2021).
- [74] Tao Wang, Jiwei Xu, Wenbo Zhang, Zeyu Gu, and Hua Zhong. “Self-adaptive cloud monitoring with online anomaly detection”. In: *Future Generation Computer Systems* 80 (Mar. 2018), pp. 89–101. ISSN: 0167739X. DOI: 10.1016/j.future.2017.09.067.
- [75] Wei Wang, Xiao Hong Guan, and Xiang Liang Zhang. “Modeling program behaviors by hidden markov models for intrusion detection”. In: *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*. Vol. 5. 2004, pp. 2830–2835. ISBN: 0780384032. DOI: 10.1109/icmlc.2004.1378514.
- [76] Yulong Wang, Qixu Wang, Xingshu Chen, Dajiang Chen, Xiaojie Fang, Mingyong Yin, and Ning Zhang. “ContainerGuard: A Real-Time Attack Detection System in Container-based Big Data Platform”. In: *IEEE Transactions on Industrial Informatics* (2020), pp. 1–1. ISSN: 1551-3203. DOI: 10.1109/TII.2020.3047416. URL: <https://ieeexplore.ieee.org/document/9309057/>.
- [77] Christina Warrender, S Forrest, and Barak A Pearlmutter. “Detecting intrusions using system calls: alternative data models”. In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)* (1999), pp. 133–145.
- [78] Roel Wieringa. “Design science as nested problem solving”. In: *Proceedings of the 4th International Conference on Design Science Research in Information*

- Systems and Technology, DESRIST '09*. New York, New York, USA: ACM Press, 2009, p. 1. ISBN: 9781605584089. DOI: 10.1145/1555619.1555630. URL: <http://portal.acm.org/citation.cfm?doid=1555619.1555630>.
- [79] Claes Wohlin. “Guidelines for snowballing in systematic literature studies and a replication in software engineering”. In: *ACM International Conference Proceeding Series*. New York, New York, USA: ACM Press, 2014. ISBN: 9781450324762. DOI: 10.1145/2601248.2601268.
 - [80] Kejiang Ye, Yangyang Liu, Guoyao Xu, and Cheng Zhong Xu. “Fault Injection and Detection for Artificial Intelligence Applications in Container-Based Clouds”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10967 LNCS. Springer Verlag, June 2018, pp. 112–127. ISBN: 9783319942940. DOI: 10.1007/978-3-319-94295-7_8.
 - [81] Dit Yan Yeung and Yuxin Ding. “Host-based intrusion detection using dynamic and static behavioral models”. In: *Pattern Recognition* 36.1 (Jan. 2003), pp. 229–243. ISSN: 00313203. DOI: 10.1016/S0031-3203(02)00026-2.
 - [82] Zhuping Zou, Yulai Xie, Kai Huang, Gongming Xu, Dan Feng, and Darrell Long. “A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest”. In: *IEEE Transactions on Cloud Computing* (Aug. 2019), pp. 1–1. ISSN: 2168-7161. DOI: 10.1109/TCC.2019.2935724. URL: <https://ieeexplore.ieee.org/document/8807263/>.

A

Appendix 1

Table A.1 presents the results for all evaluations conducted. The table uniquely identifies each evaluated technique by its run id, which is a single line of text. The run id is composed of the parameters used when evaluating the technique, and is structured in the following way:

[model]_[model_parameters]_[feature_extractor]_[feature_extractor_parameters]_none

The *_none* at the end is not used and can be ignored. For instance, the run id **iForest_100_auto_fv_100_none** can be decomposed into:

Model: **Isolation Forest**
iTrees: **100**
Contamination: **auto**
Feature extraction: **Frequency Vector**
 ΔT : **100 ms**

Table A.1: Raw results from all evaluations

Run ID	FPR	Detected %
iForest_100_auto_fv_100_none	95.45%	12.99%
iForest_100_auto_fv_200_none	100.00%	13.07%
iForest_100_auto_fv_50_none	100.00%	13.19%
iForest_100_auto_n_gram_10_none	13.64%	0.82%
iForest_100_auto_n_gram_3_none	18.18%	1.85%
iForest_100_auto_n_gram_5_none	13.64%	1.21%
iForest_100_auto_sw_10_5_none	95.45%	6.76%
iForest_100_auto_sw_11_11_none	90.91%	6.92%
iForest_100_auto_sw_11_6_none	95.45%	7.53%
iForest_100_auto_sw_23_12_none	95.45%	8.35%
iForest_100_auto_sw_7_4_none	81.82%	5.09%
iForest_100_custom_0.01_0.01_n_gram_3_none	81.82%	0.73%
iForest_100_custom_0.01_0.01_n_gram_5_none	81.82%	0.83%
iForest_100_custom_0.01_0.01_sw_10_5_none	90.91%	0.92%
iForest_100_custom_0.01_0.01_sw_11_11_none	86.36%	0.92%
iForest_100_custom_0.01_0.01_sw_11_5_none	95.45%	0.94%

Run ID	FPR	Detected %
iForest_100_custom_0.01_0.01_sw_11_6_none	90.91%	0.93%
iForest_100_custom_0.01_0.02_n_gram_3_none	86.36%	1.01%
iForest_100_custom_0.01_0.02_n_gram_5_none	86.36%	1.16%
iForest_100_custom_0.01_0.02_sw_10_5_none	90.91%	1.27%
iForest_100_custom_0.01_0.02_sw_11_11_none	86.36%	1.26%
iForest_100_custom_0.01_0.02_sw_11_5_none	95.45%	1.28%
iForest_100_custom_0.01_0.02_sw_11_6_none	90.91%	1.27%
iForest_100_custom_0.01_0.03_n_gram_3_none	86.36%	1.11%
iForest_100_custom_0.01_0.03_n_gram_5_none	86.36%	1.35%
iForest_100_custom_0.01_0.03_sw_10_5_none	90.91%	1.50%
iForest_100_custom_0.01_0.03_sw_11_11_none	86.36%	1.47%
iForest_100_custom_0.01_0.03_sw_11_5_none	95.45%	1.52%
iForest_100_custom_0.01_0.03_sw_11_6_none	90.91%	1.51%
iForest_100_custom_0.01_0.04_n_gram_3_none	86.36%	1.42%
iForest_100_custom_0.01_0.04_n_gram_5_none	86.36%	1.59%
iForest_100_custom_0.01_0.04_sw_10_5_none	90.91%	1.69%
iForest_100_custom_0.01_0.04_sw_11_11_none	86.36%	1.67%
iForest_100_custom_0.01_0.04_sw_11_5_none	95.45%	1.69%
iForest_100_custom_0.01_0.04_sw_11_6_none	90.91%	1.69%
iForest_100_custom_0.01_0.05_fv_100_none	72.73%	2.19%
iForest_100_custom_0.01_0.05_fv_200_none	63.64%	2.16%
iForest_100_custom_0.01_0.05_fv_50_none	86.36%	2.16%
iForest_100_custom_0.01_0.05_n_gram_10_none	86.36%	1.75%
iForest_100_custom_0.01_0.05_n_gram_20_none	68.18%	1.85%
iForest_100_custom_0.01_0.05_n_gram_3_none	86.36%	1.58%
iForest_100_custom_0.01_0.05_n_gram_5_none	86.36%	1.81%
iForest_100_custom_0.01_0.05_sw_10_5_none	90.91%	1.86%
iForest_100_custom_0.01_0.05_sw_11_11_none	86.36%	1.85%
iForest_100_custom_0.01_0.05_sw_11_5_none	95.45%	1.87%
iForest_100_custom_0.01_0.05_sw_11_6_none	90.91%	1.87%
iForest_100_custom_0.01_0.05_sw_23_12_none	86.36%	1.85%
iForest_100_custom_0.01_0.05_sw_7_4_none	81.82%	1.76%
iForest_100_custom_0.01_0.07_fv_100_none	77.27%	2.70%
iForest_100_custom_0.01_0.07_fv_200_none	63.64%	2.66%
iForest_100_custom_0.01_0.07_fv_50_none	86.36%	2.64%
iForest_100_custom_0.01_0.07_n_gram_10_none	86.36%	2.10%
iForest_100_custom_0.01_0.07_n_gram_20_none	68.18%	2.14%
iForest_100_custom_0.01_0.07_n_gram_3_none	86.36%	1.90%
iForest_100_custom_0.01_0.07_n_gram_5_none	86.36%	2.11%
iForest_100_custom_0.01_0.07_sw_10_5_none	90.91%	2.23%
iForest_100_custom_0.01_0.07_sw_11_11_none	86.36%	2.21%
iForest_100_custom_0.01_0.07_sw_11_5_none	95.45%	2.22%
iForest_100_custom_0.01_0.07_sw_11_6_none	90.91%	2.19%
iForest_100_custom_0.01_0.07_sw_23_12_none	86.36%	2.19%

Run ID	FPR	Detected %
iForest_100_custom_0.01_0.07_sw_7_4_none	81.82%	2.12%
iForest_100_custom_0.02_0.02_n_gram_3_none	90.91%	1.58%
iForest_100_custom_0.02_0.02_n_gram_5_none	95.45%	1.71%
iForest_100_custom_0.02_0.02_sw_10_5_none	95.45%	1.92%
iForest_100_custom_0.02_0.02_sw_11_11_none	95.45%	1.91%
iForest_100_custom_0.02_0.02_sw_11_5_none	95.45%	1.92%
iForest_100_custom_0.02_0.02_sw_11_6_none	90.91%	1.92%
iForest_100_custom_0.02_0.03_n_gram_3_none	90.91%	1.68%
iForest_100_custom_0.02_0.03_n_gram_5_none	95.45%	1.91%
iForest_100_custom_0.02_0.03_sw_10_5_none	95.45%	2.15%
iForest_100_custom_0.02_0.03_sw_11_11_none	95.45%	2.12%
iForest_100_custom_0.02_0.03_sw_11_5_none	95.45%	2.16%
iForest_100_custom_0.02_0.03_sw_11_6_none	90.91%	2.15%
iForest_100_custom_0.02_0.04_n_gram_3_none	90.91%	1.99%
iForest_100_custom_0.02_0.04_n_gram_5_none	95.45%	2.14%
iForest_100_custom_0.02_0.04_sw_10_5_none	95.45%	2.35%
iForest_100_custom_0.02_0.04_sw_11_11_none	95.45%	2.32%
iForest_100_custom_0.02_0.04_sw_11_5_none	95.45%	2.34%
iForest_100_custom_0.02_0.04_sw_11_6_none	90.91%	2.33%
iForest_100_custom_0.02_0.05_n_gram_3_none	90.91%	2.15%
iForest_100_custom_0.02_0.05_n_gram_5_none	95.45%	2.36%
iForest_100_custom_0.02_0.05_sw_10_5_none	95.45%	2.51%
iForest_100_custom_0.02_0.05_sw_11_11_none	95.45%	2.50%
iForest_100_custom_0.02_0.05_sw_11_5_none	95.45%	2.51%
iForest_100_custom_0.02_0.05_sw_11_6_none	90.91%	2.52%
iForest_100_custom_0.03_0.03_n_gram_3_none	95.45%	2.40%
iForest_100_custom_0.03_0.03_n_gram_5_none	95.45%	2.74%
iForest_100_custom_0.03_0.03_sw_10_5_none	95.45%	2.84%
iForest_100_custom_0.03_0.03_sw_11_11_none	95.45%	2.82%
iForest_100_custom_0.03_0.03_sw_11_5_none	95.45%	2.87%
iForest_100_custom_0.03_0.03_sw_11_6_none	95.45%	2.90%
iForest_100_custom_0.03_0.04_n_gram_3_none	95.45%	2.71%
iForest_100_custom_0.03_0.04_n_gram_5_none	95.45%	2.97%
iForest_100_custom_0.03_0.04_sw_10_5_none	95.45%	3.03%
iForest_100_custom_0.03_0.04_sw_11_11_none	95.45%	3.02%
iForest_100_custom_0.03_0.04_sw_11_5_none	95.45%	3.04%
iForest_100_custom_0.03_0.04_sw_11_6_none	95.45%	3.08%
iForest_100_custom_0.03_0.05_n_gram_3_none	95.45%	2.87%
iForest_100_custom_0.03_0.05_n_gram_5_none	95.45%	3.19%
iForest_100_custom_0.03_0.05_sw_10_5_none	95.45%	3.20%
iForest_100_custom_0.03_0.05_sw_11_11_none	95.45%	3.20%
iForest_100_custom_0.03_0.05_sw_11_5_none	95.45%	3.22%
iForest_100_custom_0.03_0.05_sw_11_6_none	95.45%	3.26%
iForest_100_custom_0.04_0.04_n_gram_3_none	95.45%	3.57%

Run ID	FPR	Detected %
iForest_100_custom_0.04_0.04_n_gram_5_none	95.45%	3.69%
iForest_100_custom_0.04_0.04_sw_10_5_none	95.45%	3.83%
iForest_100_custom_0.04_0.04_sw_11_11_none	95.45%	3.87%
iForest_100_custom_0.04_0.04_sw_11_5_none	95.45%	3.85%
iForest_100_custom_0.04_0.04_sw_11_6_none	95.45%	3.83%
iForest_100_custom_0.04_0.05_n_gram_3_none	95.45%	3.73%
iForest_100_custom_0.04_0.05_n_gram_5_none	95.45%	3.91%
iForest_100_custom_0.04_0.05_sw_10_5_none	95.45%	4.00%
iForest_100_custom_0.04_0.05_sw_11_11_none	95.45%	4.04%
iForest_100_custom_0.04_0.05_sw_11_5_none	95.45%	4.02%
iForest_100_custom_0.04_0.05_sw_11_6_none	95.45%	4.01%
iForest_100_custom_0.05_0.05_n_gram_3_none	95.45%	4.47%
iForest_100_custom_0.05_0.05_n_gram_5_none	95.45%	4.64%
iForest_100_custom_0.05_0.05_sw_10_5_none	95.45%	4.79%
iForest_100_custom_0.05_0.05_sw_11_11_none	95.45%	4.87%
iForest_100_custom_0.05_0.05_sw_11_5_none	95.45%	4.88%
iForest_100_custom_0.05_0.05_sw_11_6_none	95.45%	4.85%
iForest_100_exact_fv_100_none	45.45%	5.22%
iForest_100_exact_fv_200_none	36.36%	5.39%
iForest_100_exact_fv_50_none	50.00%	5.20%
iForest_100_exact_n_gram_10_none	59.09%	3.52%
iForest_100_exact_n_gram_3_none	59.09%	3.43%
iForest_100_exact_n_gram_5_none	72.73%	3.51%
iForest_100_exact_sw_10_5_none	68.18%	3.67%
iForest_100_exact_sw_11_11_none	63.64%	3.63%
iForest_100_exact_sw_11_6_none	68.18%	3.62%
iForest_100_exact_sw_23_12_none	72.73%	3.52%
iForest_100_exact_sw_7_4_none	54.55%	3.55%
iForest_150_custom_0.02_0.02_sw_10_5_none	95.45%	1.94%
iForest_150_custom_0.02_0.02_sw_11_5_none	95.45%	1.92%
iForest_150_custom_0.02_0.02_sw_11_6_none	95.45%	1.90%
iForest_150_custom_0.02_0.03_sw_10_5_none	95.45%	2.16%
iForest_150_custom_0.02_0.03_sw_11_5_none	95.45%	2.14%
iForest_150_custom_0.02_0.03_sw_11_6_none	95.45%	2.13%
iForest_150_custom_0.02_0.04_n_gram_3_none	90.91%	1.87%
iForest_150_custom_0.02_0.04_n_gram_5_none	90.91%	2.23%
iForest_150_custom_0.02_0.04_sw_10_5_none	95.45%	2.34%
iForest_150_custom_0.02_0.04_sw_11_5_none	95.45%	2.33%
iForest_150_custom_0.02_0.04_sw_11_6_none	95.45%	2.30%
iForest_150_custom_0.02_0.05_sw_10_5_none	95.45%	2.48%
iForest_150_custom_0.02_0.05_sw_11_5_none	95.45%	2.50%
iForest_150_custom_0.02_0.05_sw_11_6_none	95.45%	2.47%
iForest_150_custom_0.03_0.03_sw_10_5_none	100.00%	2.87%
iForest_150_custom_0.03_0.03_sw_11_5_none	95.45%	2.88%

Run ID	FPR	Detected %
iForest_150_custom_0.03_0.03_sw_11_6_none	95.45%	2.86%
iForest_150_custom_0.03_0.04_sw_10_5_none	100.00%	3.05%
iForest_150_custom_0.03_0.04_sw_11_5_none	95.45%	3.07%
iForest_150_custom_0.03_0.04_sw_11_6_none	95.45%	3.04%
iForest_150_custom_0.03_0.05_sw_10_5_none	100.00%	3.18%
iForest_150_custom_0.03_0.05_sw_11_5_none	95.45%	3.25%
iForest_150_custom_0.03_0.05_sw_11_6_none	95.45%	3.21%
iForest_150_custom_0.04_0.04_n_gram_3_none	95.45%	3.46%
iForest_150_custom_0.04_0.04_n_gram_5_none	95.45%	3.67%
iForest_150_custom_0.04_0.04_sw_10_5_none	100.00%	3.83%
iForest_150_custom_0.04_0.04_sw_11_5_none	95.45%	3.84%
iForest_150_custom_0.04_0.04_sw_11_6_none	100.00%	3.85%
iForest_200_auto_fv_100_none	95.45%	12.82%
iForest_200_auto_fv_200_none	100.00%	12.83%
iForest_200_auto_fv_50_none	100.00%	12.72%
iForest_200_auto_n_gram_10_none	9.09%	0.85%
iForest_200_auto_n_gram_3_none	18.18%	1.64%
iForest_200_auto_n_gram_5_none	13.64%	1.25%
iForest_200_auto_sw_10_5_none	95.45%	6.55%
iForest_200_auto_sw_11_11_none	95.45%	6.74%
iForest_200_auto_sw_11_6_none	95.45%	7.38%
iForest_200_auto_sw_23_12_none	95.45%	8.24%
iForest_200_auto_sw_7_4_none	81.82%	4.99%
iForest_200_custom_0.01_0.01_n_gram_3_none	81.82%	0.75%
iForest_200_custom_0.01_0.01_n_gram_5_none	81.82%	0.76%
iForest_200_custom_0.01_0.01_sw_10_5_none	90.91%	0.91%
iForest_200_custom_0.01_0.01_sw_11_11_none	86.36%	0.92%
iForest_200_custom_0.01_0.01_sw_11_5_none	95.45%	0.93%
iForest_200_custom_0.01_0.01_sw_11_6_none	90.91%	0.92%
iForest_200_custom_0.01_0.02_n_gram_3_none	86.36%	1.01%
iForest_200_custom_0.01_0.02_n_gram_5_none	81.82%	1.12%
iForest_200_custom_0.01_0.02_sw_10_5_none	90.91%	1.26%
iForest_200_custom_0.01_0.02_sw_11_11_none	86.36%	1.26%
iForest_200_custom_0.01_0.02_sw_11_5_none	95.45%	1.29%
iForest_200_custom_0.01_0.02_sw_11_6_none	90.91%	1.26%
iForest_200_custom_0.01_0.03_n_gram_3_none	86.36%	1.18%
iForest_200_custom_0.01_0.03_n_gram_5_none	81.82%	1.37%
iForest_200_custom_0.01_0.03_sw_10_5_none	90.91%	1.48%
iForest_200_custom_0.01_0.03_sw_11_11_none	86.36%	1.49%
iForest_200_custom_0.01_0.03_sw_11_5_none	95.45%	1.51%
iForest_200_custom_0.01_0.03_sw_11_6_none	90.91%	1.46%
iForest_200_custom_0.01_0.04_n_gram_3_none	86.36%	1.40%
iForest_200_custom_0.01_0.04_n_gram_5_none	81.82%	1.52%
iForest_200_custom_0.01_0.04_sw_10_5_none	90.91%	1.67%

Run ID	FPR	Detected %
iForest_200_custom_0.01_0.04_sw_11_11_none	86.36%	1.66%
iForest_200_custom_0.01_0.04_sw_11_5_none	95.45%	1.70%
iForest_200_custom_0.01_0.04_sw_11_6_none	90.91%	1.64%
iForest_200_custom_0.01_0.05_fv_100_none	72.73%	2.19%
iForest_200_custom_0.01_0.05_fv_200_none	63.64%	2.17%
iForest_200_custom_0.01_0.05_fv_50_none	86.36%	2.17%
iForest_200_custom_0.01_0.05_n_gram_10_none	77.27%	1.74%
iForest_200_custom_0.01_0.05_n_gram_20_none	68.18%	1.81%
iForest_200_custom_0.01_0.05_n_gram_3_none	86.36%	1.56%
iForest_200_custom_0.01_0.05_n_gram_5_none	81.82%	1.75%
iForest_200_custom_0.01_0.05_sw_10_5_none	90.91%	1.82%
iForest_200_custom_0.01_0.05_sw_11_11_none	86.36%	1.84%
iForest_200_custom_0.01_0.05_sw_11_5_none	95.45%	1.87%
iForest_200_custom_0.01_0.05_sw_11_6_none	90.91%	1.83%
iForest_200_custom_0.01_0.05_sw_23_12_none	86.36%	1.87%
iForest_200_custom_0.01_0.05_sw_7_4_none	86.36%	1.84%
iForest_200_custom_0.01_0.07_fv_100_none	77.27%	2.70%
iForest_200_custom_0.01_0.07_fv_200_none	63.64%	2.68%
iForest_200_custom_0.01_0.07_fv_50_none	86.36%	2.62%
iForest_200_custom_0.01_0.07_n_gram_10_none	77.27%	2.09%
iForest_200_custom_0.01_0.07_n_gram_20_none	68.18%	2.18%
iForest_200_custom_0.01_0.07_n_gram_3_none	86.36%	1.89%
iForest_200_custom_0.01_0.07_n_gram_5_none	81.82%	2.09%
iForest_200_custom_0.01_0.07_sw_10_5_none	90.91%	2.19%
iForest_200_custom_0.01_0.07_sw_11_11_none	86.36%	2.19%
iForest_200_custom_0.01_0.07_sw_11_5_none	95.45%	2.21%
iForest_200_custom_0.01_0.07_sw_11_6_none	90.91%	2.13%
iForest_200_custom_0.01_0.07_sw_23_12_none	86.36%	2.21%
iForest_200_custom_0.01_0.07_sw_7_4_none	86.36%	2.20%
iForest_200_custom_0.02_0.02_n_gram_3_none	90.91%	1.60%
iForest_200_custom_0.02_0.02_n_gram_5_none	90.91%	1.88%
iForest_200_custom_0.02_0.02_sw_10_5_none	95.45%	1.91%
iForest_200_custom_0.02_0.02_sw_11_11_none	95.45%	1.90%
iForest_200_custom_0.02_0.02_sw_11_5_none	95.45%	1.92%
iForest_200_custom_0.02_0.02_sw_11_6_none	95.45%	1.94%
iForest_200_custom_0.02_0.03_n_gram_3_none	90.91%	1.77%
iForest_200_custom_0.02_0.03_n_gram_5_none	90.91%	2.13%
iForest_200_custom_0.02_0.03_sw_10_5_none	95.45%	2.13%
iForest_200_custom_0.02_0.03_sw_11_11_none	95.45%	2.13%
iForest_200_custom_0.02_0.03_sw_11_5_none	95.45%	2.13%
iForest_200_custom_0.02_0.03_sw_11_6_none	95.45%	2.13%
iForest_200_custom_0.02_0.04_n_gram_3_none	90.91%	1.99%
iForest_200_custom_0.02_0.04_n_gram_5_none	90.91%	2.27%
iForest_200_custom_0.02_0.04_sw_10_5_none	95.45%	2.31%

Run ID	FPR	Detected %
iForest_200_custom_0.02_0.04_sw_11_11_none	95.45%	2.30%
iForest_200_custom_0.02_0.04_sw_11_5_none	95.45%	2.33%
iForest_200_custom_0.02_0.04_sw_11_6_none	95.45%	2.31%
iForest_200_custom_0.02_0.05_n_gram_3_none	90.91%	2.15%
iForest_200_custom_0.02_0.05_n_gram_5_none	90.91%	2.51%
iForest_200_custom_0.02_0.05_sw_10_5_none	95.45%	2.47%
iForest_200_custom_0.02_0.05_sw_11_11_none	95.45%	2.48%
iForest_200_custom_0.02_0.05_sw_11_5_none	95.45%	2.50%
iForest_200_custom_0.02_0.05_sw_11_6_none	95.45%	2.50%
iForest_200_custom_0.03_0.03_n_gram_3_none	95.45%	2.36%
iForest_200_custom_0.03_0.03_n_gram_5_none	95.45%	2.80%
iForest_200_custom_0.03_0.03_sw_10_5_none	100.00%	2.89%
iForest_200_custom_0.03_0.03_sw_11_11_none	95.45%	2.84%
iForest_200_custom_0.03_0.03_sw_11_5_none	95.45%	2.88%
iForest_200_custom_0.03_0.03_sw_11_6_none	95.45%	2.86%
iForest_200_custom_0.03_0.04_n_gram_3_none	95.45%	2.59%
iForest_200_custom_0.03_0.04_n_gram_5_none	95.45%	2.94%
iForest_200_custom_0.03_0.04_sw_10_5_none	100.00%	3.07%
iForest_200_custom_0.03_0.04_sw_11_11_none	95.45%	3.00%
iForest_200_custom_0.03_0.04_sw_11_5_none	95.45%	3.08%
iForest_200_custom_0.03_0.04_sw_11_6_none	95.45%	3.04%
iForest_200_custom_0.03_0.05_n_gram_3_none	95.45%	2.75%
iForest_200_custom_0.03_0.05_n_gram_5_none	95.45%	3.18%
iForest_200_custom_0.03_0.05_sw_10_5_none	100.00%	3.23%
iForest_200_custom_0.03_0.05_sw_11_11_none	95.45%	3.19%
iForest_200_custom_0.03_0.05_sw_11_5_none	95.45%	3.24%
iForest_200_custom_0.03_0.05_sw_11_6_none	95.45%	3.23%
iForest_200_custom_0.04_0.04_n_gram_3_none	95.45%	3.48%
iForest_200_custom_0.04_0.04_n_gram_5_none	95.45%	3.66%
iForest_200_custom_0.04_0.04_sw_10_5_none	100.00%	3.85%
iForest_200_custom_0.04_0.04_sw_11_11_none	95.45%	3.82%
iForest_200_custom_0.04_0.04_sw_11_5_none	95.45%	3.86%
iForest_200_custom_0.04_0.04_sw_11_6_none	95.45%	3.84%
iForest_200_custom_0.04_0.05_n_gram_3_none	95.45%	3.64%
iForest_200_custom_0.04_0.05_n_gram_5_none	95.45%	3.89%
iForest_200_custom_0.04_0.05_sw_10_5_none	100.00%	4.00%
iForest_200_custom_0.04_0.05_sw_11_11_none	95.45%	4.00%
iForest_200_custom_0.04_0.05_sw_11_5_none	95.45%	4.03%
iForest_200_custom_0.04_0.05_sw_11_6_none	95.45%	4.03%
iForest_200_custom_0.05_0.05_n_gram_3_none	95.45%	4.33%
iForest_200_custom_0.05_0.05_n_gram_5_none	95.45%	4.73%
iForest_200_custom_0.05_0.05_sw_10_5_none	100.00%	4.81%
iForest_200_custom_0.05_0.05_sw_11_11_none	100.00%	4.85%
iForest_200_custom_0.05_0.05_sw_11_5_none	100.00%	4.85%

Run ID	FPR	Detected %
iForest_200_custom_0.05_0.05_sw_11_6_none	100.00%	4.84%
iForest_200_exact_fv_100_none	45.45%	5.19%
iForest_200_exact_fv_200_none	36.36%	5.39%
iForest_200_exact_fv_50_none	50.00%	5.19%
iForest_200_exact_n_gram_10_none	54.55%	3.55%
iForest_200_exact_n_gram_3_none	68.18%	3.35%
iForest_200_exact_n_gram_5_none	59.09%	3.50%
iForest_200_exact_sw_10_5_none	68.18%	3.65%
iForest_200_exact_sw_11_11_none	59.09%	3.59%
iForest_200_exact_sw_11_6_none	72.73%	3.55%
iForest_200_exact_sw_23_12_none	63.64%	3.57%
iForest_200_exact_sw_7_4_none	54.55%	3.67%
iForest_250_custom_0.02_0.02_sw_10_5_none	95.45%	1.92%
iForest_250_custom_0.02_0.02_sw_11_5_none	95.45%	1.92%
iForest_250_custom_0.02_0.02_sw_11_6_none	95.45%	1.92%
iForest_250_custom_0.02_0.03_sw_10_5_none	95.45%	2.14%
iForest_250_custom_0.02_0.03_sw_11_5_none	95.45%	2.15%
iForest_250_custom_0.02_0.03_sw_11_6_none	95.45%	2.13%
iForest_250_custom_0.02_0.04_n_gram_3_none	90.91%	1.90%
iForest_250_custom_0.02_0.04_n_gram_5_none	86.36%	2.10%
iForest_250_custom_0.02_0.04_sw_10_5_none	95.45%	2.32%
iForest_250_custom_0.02_0.04_sw_11_5_none	95.45%	2.33%
iForest_250_custom_0.02_0.04_sw_11_6_none	95.45%	2.31%
iForest_250_custom_0.02_0.05_sw_10_5_none	95.45%	2.45%
iForest_250_custom_0.02_0.05_sw_11_5_none	95.45%	2.50%
iForest_250_custom_0.02_0.05_sw_11_6_none	95.45%	2.49%
iForest_250_custom_0.03_0.03_sw_10_5_none	95.45%	2.86%
iForest_250_custom_0.03_0.03_sw_11_5_none	95.45%	2.85%
iForest_250_custom_0.03_0.03_sw_11_6_none	95.45%	2.90%
iForest_250_custom_0.03_0.04_n_gram_5_none	95.45%	2.94%
iForest_250_custom_0.03_0.04_sw_10_5_none	95.45%	3.05%
iForest_250_custom_0.03_0.04_sw_11_5_none	95.45%	3.03%
iForest_250_custom_0.03_0.04_sw_11_6_none	95.45%	3.09%
iForest_250_custom_0.03_0.05_n_gram_3_none	95.45%	2.84%
iForest_250_custom_0.03_0.05_n_gram_5_none	95.45%	3.17%
iForest_250_custom_0.03_0.05_sw_10_5_none	95.45%	3.18%
iForest_250_custom_0.03_0.05_sw_11_5_none	95.45%	3.20%
iForest_250_custom_0.03_0.05_sw_11_6_none	95.45%	3.27%
iForest_250_custom_0.04_0.04_n_gram_3_none	95.45%	3.37%
iForest_250_custom_0.04_0.04_n_gram_5_none	95.45%	3.63%
iForest_250_custom_0.04_0.04_sw_10_5_none	100.00%	3.84%
iForest_250_custom_0.04_0.04_sw_11_5_none	95.45%	3.87%
iForest_250_custom_0.04_0.04_sw_11_6_none	95.45%	3.83%
iForest_50_auto_fv_100_none	95.45%	13.04%

Run ID	FPR	Detected %
iForest_50_auto_fv_200_none	100.00%	12.95%
iForest_50_auto_fv_50_none	100.00%	12.98%
iForest_50_auto_n_gram_10_none	13.64%	0.94%
iForest_50_auto_n_gram_3_none	18.18%	2.16%
iForest_50_auto_n_gram_5_none	18.18%	1.38%
iForest_50_auto_sw_10_5_none	95.45%	6.98%
iForest_50_auto_sw_11_11_none	95.45%	8.45%
iForest_50_auto_sw_11_6_none	100.00%	7.31%
iForest_50_auto_sw_23_12_none	95.45%	8.35%
iForest_50_auto_sw_7_4_none	86.36%	6.14%
iForest_50_custom_0.01_0.01_n_gram_3_none	77.27%	0.71%
iForest_50_custom_0.01_0.01_n_gram_5_none	90.91%	0.81%
iForest_50_custom_0.01_0.01_sw_10_5_none	90.91%	0.92%
iForest_50_custom_0.01_0.01_sw_11_11_none	90.91%	0.92%
iForest_50_custom_0.01_0.01_sw_11_5_none	95.45%	0.93%
iForest_50_custom_0.01_0.01_sw_11_6_none	95.45%	0.94%
iForest_50_custom_0.01_0.02_n_gram_3_none	81.82%	0.97%
iForest_50_custom_0.01_0.02_n_gram_5_none	90.91%	1.22%
iForest_50_custom_0.01_0.02_sw_10_5_none	90.91%	1.26%
iForest_50_custom_0.01_0.02_sw_11_11_none	90.91%	1.27%
iForest_50_custom_0.01_0.02_sw_11_5_none	95.45%	1.27%
iForest_50_custom_0.01_0.02_sw_11_6_none	95.45%	1.27%
iForest_50_custom_0.01_0.03_n_gram_3_none	81.82%	1.16%
iForest_50_custom_0.01_0.03_n_gram_5_none	90.91%	1.39%
iForest_50_custom_0.01_0.03_sw_10_5_none	90.91%	1.50%
iForest_50_custom_0.01_0.03_sw_11_11_none	90.91%	1.49%
iForest_50_custom_0.01_0.03_sw_11_5_none	95.45%	1.51%
iForest_50_custom_0.01_0.03_sw_11_6_none	95.45%	1.51%
iForest_50_custom_0.01_0.04_n_gram_3_none	81.82%	1.39%
iForest_50_custom_0.01_0.04_n_gram_5_none	90.91%	1.59%
iForest_50_custom_0.01_0.04_sw_10_5_none	90.91%	1.68%
iForest_50_custom_0.01_0.04_sw_11_11_none	90.91%	1.67%
iForest_50_custom_0.01_0.04_sw_11_5_none	95.45%	1.69%
iForest_50_custom_0.01_0.04_sw_11_6_none	95.45%	1.67%
iForest_50_custom_0.01_0.05_fv_100_none	72.73%	2.19%
iForest_50_custom_0.01_0.05_fv_200_none	59.09%	2.16%
iForest_50_custom_0.01_0.05_fv_50_none	86.36%	2.17%
iForest_50_custom_0.01_0.05_n_gram_10_none	86.36%	1.82%
iForest_50_custom_0.01_0.05_n_gram_20_none	77.27%	1.75%
iForest_50_custom_0.01_0.05_n_gram_3_none	81.82%	1.48%
iForest_50_custom_0.01_0.05_n_gram_5_none	90.91%	1.73%
iForest_50_custom_0.01_0.05_sw_10_5_none	90.91%	1.86%
iForest_50_custom_0.01_0.05_sw_11_11_none	90.91%	1.85%
iForest_50_custom_0.01_0.05_sw_11_5_none	95.45%	1.87%

Run ID	FPR	Detected %
iForest_50_custom_0.01_0.05_sw_11_6_none	95.45%	1.87%
iForest_50_custom_0.01_0.05_sw_23_12_none	86.36%	1.87%
iForest_50_custom_0.01_0.05_sw_7_4_none	81.82%	1.83%
iForest_50_custom_0.01_0.07_fv_100_none	77.27%	2.69%
iForest_50_custom_0.01_0.07_fv_200_none	63.64%	2.65%
iForest_50_custom_0.01_0.07_fv_50_none	86.36%	2.64%
iForest_50_custom_0.01_0.07_n_gram_10_none	86.36%	2.14%
iForest_50_custom_0.01_0.07_n_gram_20_none	77.27%	2.11%
iForest_50_custom_0.01_0.07_n_gram_3_none	81.82%	1.89%
iForest_50_custom_0.01_0.07_n_gram_5_none	90.91%	2.16%
iForest_50_custom_0.01_0.07_sw_10_5_none	90.91%	2.16%
iForest_50_custom_0.01_0.07_sw_11_11_none	90.91%	2.20%
iForest_50_custom_0.01_0.07_sw_11_6_none	95.45%	2.20%
iForest_50_custom_0.01_0.07_sw_23_12_none	86.36%	2.20%
iForest_50_custom_0.01_0.07_sw_7_4_none	81.82%	2.12%
iForest_50_custom_0.02_0.02_n_gram_3_none	90.91%	1.48%
iForest_50_custom_0.02_0.02_n_gram_5_none	95.45%	1.85%
iForest_50_custom_0.02_0.02_sw_10_5_none	95.45%	1.94%
iForest_50_custom_0.02_0.02_sw_11_11_none	95.45%	1.92%
iForest_50_custom_0.02_0.02_sw_11_5_none	95.45%	1.88%
iForest_50_custom_0.02_0.02_sw_11_6_none	95.45%	1.92%
iForest_50_custom_0.02_0.03_n_gram_3_none	90.91%	1.67%
iForest_50_custom_0.02_0.03_n_gram_5_none	95.45%	2.02%
iForest_50_custom_0.02_0.03_sw_10_5_none	95.45%	2.18%
iForest_50_custom_0.02_0.03_sw_11_11_none	95.45%	2.14%
iForest_50_custom_0.02_0.03_sw_11_5_none	95.45%	2.13%
iForest_50_custom_0.02_0.03_sw_11_6_none	95.45%	2.16%
iForest_50_custom_0.02_0.04_n_gram_3_none	90.91%	1.90%
iForest_50_custom_0.02_0.04_n_gram_5_none	95.45%	2.21%
iForest_50_custom_0.02_0.04_sw_10_5_none	95.45%	2.36%
iForest_50_custom_0.02_0.04_sw_11_11_none	95.45%	2.32%
iForest_50_custom_0.02_0.04_sw_11_5_none	95.45%	2.31%
iForest_50_custom_0.02_0.04_sw_11_6_none	95.45%	2.33%
iForest_50_custom_0.02_0.05_n_gram_3_none	90.91%	1.99%
iForest_50_custom_0.02_0.05_n_gram_5_none	95.45%	2.35%
iForest_50_custom_0.02_0.05_sw_10_5_none	95.45%	2.54%
iForest_50_custom_0.02_0.05_sw_11_11_none	95.45%	2.50%
iForest_50_custom_0.02_0.05_sw_11_5_none	95.45%	2.48%
iForest_50_custom_0.02_0.05_sw_11_6_none	95.45%	2.52%
iForest_50_custom_0.03_0.03_n_gram_3_none	95.45%	2.56%
iForest_50_custom_0.03_0.03_n_gram_5_none	95.45%	2.63%
iForest_50_custom_0.03_0.03_sw_10_5_none	95.45%	2.91%
iForest_50_custom_0.03_0.03_sw_11_11_none	95.45%	2.88%
iForest_50_custom_0.03_0.03_sw_11_5_none	95.45%	2.87%

Run ID	FPR	Detected %
iForest_50_custom_0.03_0.03_sw_11_6_none	100.00%	2.85%
iForest_50_custom_0.03_0.04_n_gram_3_none	95.45%	2.79%
iForest_50_custom_0.03_0.04_n_gram_5_none	95.45%	2.83%
iForest_50_custom_0.03_0.04_sw_10_5_none	95.45%	3.09%
iForest_50_custom_0.03_0.04_sw_11_11_none	95.45%	3.06%
iForest_50_custom_0.03_0.04_sw_11_5_none	95.45%	3.05%
iForest_50_custom_0.03_0.04_sw_11_6_none	100.00%	3.02%
iForest_50_custom_0.03_0.05_n_gram_3_none	95.45%	2.88%
iForest_50_custom_0.03_0.05_n_gram_5_none	95.45%	2.97%
iForest_50_custom_0.03_0.05_sw_10_5_none	95.45%	3.27%
iForest_50_custom_0.03_0.05_sw_11_11_none	95.45%	3.25%
iForest_50_custom_0.03_0.05_sw_11_5_none	95.45%	3.22%
iForest_50_custom_0.03_0.05_sw_11_6_none	100.00%	3.21%
iForest_50_custom_0.04_0.04_n_gram_3_none	95.45%	3.58%
iForest_50_custom_0.04_0.04_n_gram_5_none	95.45%	3.77%
iForest_50_custom_0.04_0.04_sw_10_5_none	100.00%	3.84%
iForest_50_custom_0.04_0.04_sw_11_11_none	95.45%	3.88%
iForest_50_custom_0.04_0.04_sw_11_5_none	100.00%	3.87%
iForest_50_custom_0.04_0.04_sw_11_6_none	100.00%	3.88%
iForest_50_custom_0.04_0.05_n_gram_3_none	95.45%	3.67%
iForest_50_custom_0.04_0.05_n_gram_5_none	95.45%	3.91%
iForest_50_custom_0.04_0.05_sw_10_5_none	100.00%	4.02%
iForest_50_custom_0.04_0.05_sw_11_11_none	95.45%	4.06%
iForest_50_custom_0.04_0.05_sw_11_5_none	100.00%	4.04%
iForest_50_custom_0.04_0.05_sw_11_6_none	100.00%	4.07%
iForest_50_custom_0.05_0.05_n_gram_3_none	95.45%	4.49%
iForest_50_custom_0.05_0.05_n_gram_5_none	95.45%	4.62%
iForest_50_custom_0.05_0.05_sw_10_5_none	100.00%	4.88%
iForest_50_custom_0.05_0.05_sw_11_11_none	95.45%	4.83%
iForest_50_custom_0.05_0.05_sw_11_5_none	100.00%	4.86%
iForest_50_custom_0.05_0.05_sw_11_6_none	100.00%	4.87%
iForest_50_exact_fv_100_none	50.00%	5.17%
iForest_50_exact_fv_200_none	36.36%	5.40%
iForest_50_exact_fv_50_none	50.00%	5.22%
iForest_50_exact_n_gram_10_none	63.64%	3.55%
iForest_50_exact_n_gram_3_none	54.55%	3.51%
iForest_50_exact_n_gram_5_none	59.09%	3.59%
iForest_50_exact_sw_10_5_none	63.64%	3.62%
iForest_50_exact_sw_11_11_none	59.09%	3.59%
iForest_50_exact_sw_11_6_none	59.09%	3.47%
iForest_50_exact_sw_23_12_none	68.18%	3.51%
iForest_50_exact_sw_7_4_none	63.64%	3.57%
iForest_500_custom_0.01_0.05_n_gram_20_none	68.18%	1.82%
iForest_500_custom_0.01_0.05_n_gram_5_none	86.36%	1.72%

Run ID	FPR	Detected %
iForest_500_custom_0.01_0.05_sw_10_5_none	95.45%	1.80%
iForest_500_custom_0.01_0.05_sw_11_11_none	86.36%	1.86%
iForest_500_custom_0.01_0.05_sw_11_5_none	95.45%	1.87%
iForest_500_custom_0.01_0.05_sw_11_6_none	95.45%	1.86%
iForest_500_custom_0.01_0.07_n_gram_20_none	68.18%	2.18%
iForest_500_custom_0.01_0.07_n_gram_5_none	86.36%	2.08%
iForest_500_custom_0.01_0.07_sw_10_5_none	95.45%	2.14%
iForest_500_custom_0.01_0.07_sw_11_11_none	86.36%	2.17%
iForest_500_custom_0.01_0.07_sw_11_5_none	95.45%	2.17%
iForest_500_custom_0.01_0.07_sw_11_6_none	95.45%	2.17%
iForest_500_custom_0.02_0.02_sw_10_5_none	95.45%	1.93%
iForest_500_custom_0.02_0.02_sw_11_5_none	95.45%	1.92%
iForest_500_custom_0.02_0.02_sw_11_6_none	95.45%	1.91%
iForest_500_custom_0.02_0.03_sw_10_5_none	95.45%	2.16%
iForest_500_custom_0.02_0.03_sw_11_5_none	95.45%	2.15%
iForest_500_custom_0.02_0.03_sw_11_6_none	95.45%	2.14%
iForest_500_custom_0.02_0.04_n_gram_3_none	90.91%	1.89%
iForest_500_custom_0.02_0.04_sw_10_5_none	95.45%	2.32%
iForest_500_custom_0.02_0.04_sw_11_5_none	95.45%	2.33%
iForest_500_custom_0.02_0.04_sw_11_6_none	95.45%	2.33%
iForest_500_custom_0.02_0.05_sw_10_5_none	95.45%	2.46%
iForest_500_custom_0.02_0.05_sw_11_5_none	95.45%	2.51%
iForest_500_custom_0.02_0.05_sw_11_6_none	95.45%	2.50%
iForest_500_custom_0.03_0.03_sw_10_5_none	95.45%	2.89%
iForest_500_custom_0.03_0.03_sw_11_5_none	95.45%	2.91%
iForest_500_custom_0.03_0.03_sw_11_6_none	95.45%	2.87%
iForest_500_custom_0.03_0.04_n_gram_3_none	95.45%	2.69%
iForest_500_custom_0.03_0.04_n_gram_5_none	95.45%	2.83%
iForest_500_custom_0.03_0.04_sw_10_5_none	95.45%	3.05%
iForest_500_custom_0.03_0.04_sw_11_5_none	95.45%	3.09%
iForest_500_custom_0.03_0.04_sw_11_6_none	95.45%	3.06%
iForest_500_custom_0.03_0.05_n_gram_3_none	95.45%	2.82%
iForest_500_custom_0.03_0.05_n_gram_5_none	95.45%	2.99%
iForest_500_custom_0.03_0.05_sw_10_5_none	95.45%	3.19%
iForest_500_custom_0.03_0.05_sw_11_5_none	95.45%	3.26%
iForest_500_custom_0.03_0.05_sw_11_6_none	95.45%	3.24%
iForest_500_custom_0.04_0.04_n_gram_3_none	95.45%	3.53%
iForest_500_custom_0.04_0.04_n_gram_5_none	95.45%	3.70%
iForest_500_custom_0.04_0.04_sw_10_5_none	100.00%	3.83%
iForest_500_custom_0.04_0.04_sw_11_5_none	100.00%	3.84%
iForest_500_custom_0.04_0.04_sw_11_6_none	95.45%	3.84%