UNIVERSIDADE D
COIMBRA

José Eduardo Ferreira Flora

CONTAINER-LEVEL INTRUSION DETECTION FOR MULTI-TENANT ENVIRONMENTS

Dissertation in the context of the Master in Informatics Security
advised by Professor Dr. Nuno Antunes and presented to
Faculty of Sciences and Technology / Department of Informatics Engineering

September 2019

Faculty of Sciences and Tecnology
Department of Informatics Engineering

Container-level Intrusion Detection for Multi-tenant Environments

José Eduardo Ferreira Flora

Dissertation in the context of the Master in Informatics Security
advised by Prof. Nuno Antunes and presented to the
Faculty of Sciences and Technology / Department of Informatics Engineering.

September 2019

UNIVERSIDADE D
COIMBRA

This work is within the informatics security specialization area and was carried out in the Software and Systems Engineering (SSE) Group of the Centre for Informatics and Systems of the University of Coimbra (CISUC).

This work is partially supported by the project METRICS (POCI-01-0145-FEDER-032504), co-funded by the Portuguese Foundation for Science and Technology (FCT) and by the *Fundo Europeu de Desenvolvimento Regional* (FEDER) through *Portugal 2020 - Programa Operacional Competitividade e Internacionalização (POCI).*
It is also partially supported by the project ATMOSPHERE, funded by the Brazilian Ministry of Science, Technology and Innovation (51119 - MCTI/RNP 4th Coordinated Call) and by the European Commission under the Cooperation Programme, H2020 grant agreement no 777154.

This work has been supervised by Professor Nuno Manuel dos Santos Antunes, Assistant Professor at the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

This page is intentionally left blank.

Acknowledgements

First and foremost, I would like to thank my advisor Professor Nuno Antunes for providing me with this opportunity. For his patience when I made silly mistakes, and even when I did not, his wisdom and his willingness and readiness to share with me enlightening thoughts, perspectives and advice for both work and personal matters and for some reason believing in me throughout the last almost three years.

In addition, I would like to thank my lab colleagues, specially Nuno Cardoso, Rui Silva, José Pereira and Inês Valentim for always keeping the mood at the highest and helping to make the day more pleasant and enjoyable even when things went sideways. For all the enormous fun and enjoyment during afternoons playing video games, when sometimes we should be working, and all the stress-breaking after lunch domino matches, those were very engaging and fun moments.

Furthermore, I would like to thank Ana, although I have not enough words to express what her presence in my life means, and how this could not have been possible without her emotional support, care and everyday love, I would say that it is like salt to our food. Thank you for making me see reason when I was stressed, doubtful or just lazy. I hope our connection remains with great love and care for a long, long time as it has until now.

Last but not least, I would like to thank my parents all their support and all the sacrifices they have made throughout their life in order to make this possible, I would not have been able to achieve it otherwise. Even though they do not fully comprehend what I really do, I know it makes them very happy and proud each time I achieve my goals. Also, my sisters a warm thanks for the support and care, for aiding in my development as a person and making me who I am today, and for always rooting to see me thrive and be happy as well.

This page is intentionally left blank.

Abstract

Cloud computing provides a convenient, on-demand, elastic and ubiquitous service enabled by high-performance virtualisation systems among other features. There has been a growth in containers' use over the last years, even in business-critical scenarios. Their lightweight, easier and more efficient instantiation empowers not only an elastic and on-demand use of resources, but also a straightforward resource administration. However, the adoption of containers also increases **security risks exacerbated by multi-tenant environments** that should be carefully analysed. Attacks led by other tenants present in the same infrastructure are particularly concerning, and counter-measures such as intrusion detection systems should be deployed at container-level.

In this work, we study the effectiveness and applicability of intrusion detection techniques in container-based multi-tenant systems. For this, we **propose a methodology based on attack injection that uses representative workloads and attacks exploits in a container-based system**, to generate the traces required to train and test the classifiers. Following the proposed methodology, we devised an experimental campaign to evaluate three state-of-the-art and widely used intrusion detection algorithms: BoSC, STIDE, and HMM. A version of MariaDB with known vulnerabilities was deployed into containers, and submitted to two variations of the TPC-C workload. In each attack slot, one of five diverse attacks was executed following the attack injection procedure. The experiment was also performed in a traditional OS setup, to study advantages and drawbacks.

The results of the experimental campaign indicate that the algorithms are applicable in this domain since several configurations obtained very good results in all scenarios: in Docker setups (recall `>=0.98`, precision `>=0.72`), and slightly worse, but yet satisfactory in LXC. The observed results for the OS setup were worse, indicating that difficulty of properly defining the monitoring surface. Our results indicate that **anomaly-based intrusion detection is effective in this environment**, leading the way to the application of other security techniques such as intrusion tolerance.

Keywords

Cloud Security, Security Evaluation, Intrusion Detection, Containers, Anomaly Detection

This page is intentionally left blank.

Resumo

A computação em nuvem fornece um serviço conveniente, *on-demand*, elástico e ubíquo, através de sistemas de virtualização de alto desempenho, entre outros recursos. Houve um crescimento no uso de *containers* nos últimos anos, mesmo em cenários críticos para o negócio. A sua instanciação leve, fácil e eficiente permite não apenas um uso elástico e *on-demand* de recursos, mas também uma administração facilitada. No entanto, a adoção de *containers* também **aumenta os riscos de segurança agravados pelos ambientes de *multi-tenancy*** que devem ser analisados cuidadosamente. Os ataques realizados por outros inquilinos presentes na mesma infraestrutura são particularmente preocupantes e medidas preventivas, como sistemas de detecção de intrusão, devem ser implementadas ao nível dos *containers*.

Neste trabalho, estudamos a eficácia e a aplicabilidade de técnicas de detecção de intrusão em sistemas *multi-tenant* baseados em *containers*. Para isso, **propomos uma metodologia baseada na injeção de ataques que utiliza cargas de trabalho representativas e exploração de ataques num sistema baseado em *containers***, para gerar os dados necessários para treinar e testar os classificadores. Seguindo a metodologia proposta, desenvolvemos uma campanha experimental para avaliar três algoritmos de detecção de intrusões de estado da arte e amplamente utilizados: BoSC, STIDE e HMM. Uma versão do MariaDB com vulnerabilidades conhecidas foi colocada em *containers* e submetida a duas variações da carga de trabalho do TPC-C. Em cada slot de ataque, um de cinco ataques representativos foi executado de acordo com o procedimento de injeção do ataque. A experiência também foi realizada numa configuração tradicional do sistema operativo, para estudar vantagens e desvantagens.

Os resultados do trabalho experimental indicam que os algoritmos são aplicáveis neste domínio, pois várias configurações obtiveram muito bons resultados em todos os cenários: nas configurações do Docker (*recall* `>=0,98`, *precision* `>=0,72`) e um pouco pior, mas ainda satisfatório no LXC. Os resultados observados para a configuração do SO foram piores, indicando a dificuldade de definir adequadamente a superfície de monitorização. Os nossos resultados indicam que a **detecção de intrusões com base em anomalias é eficaz neste ambiente**, abrindo caminho para a aplicação de outras técnicas de segurança, como tolerância a intrusões.

Palavras-Chave

Segurança na Nuvem, Avaliação de Segurança, Detecção de Intrusão, Containers, Detecção de Anomalias

This page is intentionally left blank.

Contents

List of Abbreviations

ANN Artificial Neural Network. 12, 17

BoSC Bags of System Calls. 3, 12, 13, 17, 28, 30, 34, 35, 37, 38, 40–43, 49, 53, 54, 56, 58–72

CVE Common Vulnerabilities and Exposure. 51

CVSS Common Vulnerability Scoring System. 52

DBMS Database Management System. 3, 4, 28, 39, 51, 52

FN False Negative. 53, 54

FP False Positive. 53, 54

HIDS Host-based Intrusion Detection System. 9, 11

HMM Hidden Markov Models. 3, 12, 17, 35, 49, 53, 54, 56, 58–61, 63, 64, 66–68, 70, 72

IDS Intrusion Detection System. 2, 7, 9–14, 16–20, 22, 28, 52

KNN K-Nearest Neighbour. 13, 17, 76

KVM Kernel-based Virtual Machine. 38, 54

MEC Multi-access Edge Computing. 1

ML Machine Learning. 13

NB Naïve Bayes. 13, 17, 76

OCSVM One-Class Support Vector Machines. 13, 17, 76

OS Operating System. 1–3, 9, 10, 22, 24, 27, 29, 33, 34, 38, 55, 56, 75

PoC Proof of Concept. 45, 51, 52

ROC Receiver Operating Characteristic. 19, 63, 64

STIDE Sequence Time-Delaying Embedding. 3, 13, 17, 34, 35, 37, 38, 40–43, 49, 53, 54, 56, 58–68, 70–72

SVM Support Vector Machines. 13, 17

TN True Negative. 53

TP True Positive. 53, 54

UNM University of New Mexico. 28

VM Virtual Machine. 1, 2, 22, 29, 32

List of Figures

List of Tables

This page is intentionally left blank.

List of Publications

This dissertation is partially based on the work presented in the following publication:

- **José Flora**, Nuno Antunes, "Studying the Applicability of Intrusion Detection to Multi-tenant Container Environments", *15th European Dependable Computing Conference (EDCC 2019)*, **short paper**, Naples, Italy, September 17-20, 2019.
  - ***Abstract:*** *The use of containers in cloud-based applications allows for rapid and scalable deployments. Containers are lightweight and appealing to be used even in business-critical systems, but their use implies great security concerns, which are exacerbated in multi-tenant environments. To mitigate these concerns, techniques such as intrusion detection are a must, however, in the containers' context, it has received limited attention. Thus, it is necessary to define an improved approach to container-level intrusion detection for multi-tenant environments. In this paper we make a preliminary feasibility analysis of host-based container-level intrusion detection. For this, we are currently focusing on achieving a stable container profile definition and the results obtained show we are following the correct path.*

This page is intentionally left blank.

Chapter 1

Introduction

Cloud computing provides its users a convenient, **on-demand**, **elastic** and **ubiquitous** service enabled by fast wide-area networks, powerful computer servers and high-performance virtualisation systems [1]. Virtualisation allows physical resources to be shared by multiple clients of the same cloud service provider through the emulation and isolation of resources for each customer.

There are two main types of virtualisation: *hardware virtualisation* and *operating system-level virtualisation* [2]. While **hardware virtualisation** consists in the emulation of hardware for each Operating System (OS) to interact with, OS virtualisation consists in the emulation of an OS which share the same OS kernel [2]. The virtualisation of hardware can be achieved in two forms, either in bare metal virtualisation or hosted virtualisation. Bare metal virtualisation consists in the direct deployment of the hypervisor into the hardware, whereas for hosted virtualisation, the hypervisor runs on top of the OS [3]. On the other hand, **OS-level virtualisation** consist of an abstraction of the application layer through resource isolation [2]. This isolation is possible due to the use of mechanisms present in the Linux kernel, such as *control groups* [4] and *namespaces* [5]. As a result of these features, it is possible to execute multiple instances, each one using a set of resources on a single kernel of a host machine.

Although initially Virtual Machines (VMs) became more popular, there has been a growth in containers' use over the last years [6]. This usage and popularity growth is mostly due to the release of Docker [7], which is a containerisation platform that has been released in 2013. Moreover, the fact that containers are lightweight facilitates its adoption. Containers have an easier and more efficient instantiation thereby empowering the **elastic** and **on-demand** use of resources. Besides, from a management perspective, containers are also appealing due to easier resource administration owning to its lighter-weight [8].

Containers have the potential to unlock and/or improve many technological solutions ranging from Mixed-Criticality Systems (MCS) [9] to Multi-access Edge Computing (MEC) [10], among others. As an example, MEC consists in bringing the cloud computing power closer to the end-users, and provides certain types of services with near real-time characteristics [10]. Consequently, containers allow an efficient use of the limited computing power on each MEC device. However, the adoption of containers also increases security risks that should be carefully analysed [2]. MEC services may contain sensitive information,

and therefore should be carefully evaluated due to data confidentiality and integrity problems. This is even more concerning when we consider multi-tenancy: in practice, different application vendors may be running on the same infrastructure, raising concerns of data separation and performance isolation [10].

The **security concerns raised by containers adoption are thus exacerbated in multi-tenant systems** [2, 9, 10]. These tenants, which acquired the resources lawfully, are in some occasions competing for resources. They may try to perform malicious actions towards its neighbours or the host itself, which may lead to harmful consequences when dealing with, e.g., business-critical systems. Vulnerabilities present in the containers runtime software can cause severe damage if they permit *"container escape"* scenarios, where malicious software can attack resources of neighbours containers or the host OS itself [2]. This compromising could allow an attacker to explore and access other containers or monitor their communications. Thus, these vulnerabilities may compromise the integrity, availability or confidentiality of containers within the infrastructure, reason why we selected the attack injection procedure as a way to understand whether intrusion detection is an applicable counter-measure to these environments.

In this scenario, it is of utmost importance to put in place counter-measures as a way to mitigate or, at least, reduce the risk which tenants are exposed to. In this work, **we are particularly concerned with attacks led by other tenants present in the same container-based infrastructure**, and we argue that counter-measures systems should be deployed at container-level. In particular, intrusion detection systems may have a very important role in the mitigation of these threats.

Intrusion Detection Systems (IDSes) are one of the most applied security measures to cloud deployments. Generally, IDSes are deployed at network-level and at host-level. While network-based IDSes monitor and inspect the packets in transit on a network, host-based IDSes focus on user/application monitoring on a designated machine [11]. In this work, we are particularly interested on host-based IDSes since they are effective on cloud infrastructures [12]. For instance, the use of sequences of system calls to detect intrusions at host-level is a methodology introduced in 1996 [13] that is still effective and widely-used. Other approaches, such as neural networks and statistical models, have also been used [14].

Regarding cloud intrusion detection, there has also been some developments in terms of deployment and monitoring models, for instance distributed IDSes were proposed as a way to monitor a large and heterogeneous network of computers [15], which is the case of a cloud infrastructure. Although there are great advances in terms of intrusion detection systems for VMs, **the contributions for container-based systems approaches are limited**. In practice, the existing works are incomplete and hard to generalise, mainly because the used workloads and/or attacks have substantial issues of representativeness [16, 17].

Several techniques have been proposed for the the evaluation of IDSes [18] in terms of their ability in detecting attacks. Usually, traces or workloads including malicious activity are a requirement to perform this evaluation [18]. However, the generation of such traces is challenging, and to the best of our knowledge, currently there are no publicly available system call traces from attacks to container-based systems [16]. **Attack injection techniques are one possible way to mitigate this issue** [19]. This method consists in the controlled execution of vulnerability exploitation as a way to produce data to feed the

analyser in order to observe its reaction [20]. It has been successfully applied in the past to web applications [21] and hypervisors [11, 19], and can help in our efforts as long as we have representative workloads and we are able to devise or reuse effective attacks.

In this work, we **study the effectiveness and applicability of intrusion detection techniques in container-based multi-tenant systems**. For this, we propose a methodology based on attack injection concepts, that uses representative workloads and attacks exploits in a representative container-based system to generate the traces required for training classifiers and testing them in order to obtain results. This methodology was conceived based on the results of the preliminary analysis performed. These results showed the capacity of Sequence Time-Delaying Embedding (STIDE) and Bags of System Calls (BoSC) to define a stable profile for each containerisation platform revealing, as expected, a faster definition of a stable profile when BoSC is used. The preliminary analysis also allowed us to define the monitoring surface and to select the monitoring tool to be used.

Following this methodology, we devised an **experimental campaign to evaluate three state-of-the-art and widely used intrusion detection algorithms** in a container-based in multi-tenant environment. For this, we adapted the deployment of a version of the Database Management System (DBMS) MariaDB with known vulnerabilities to application and OS containers and then we used two variations of the TPC-C benchmark as workload, and performed an attack injection procedure using five different representative attacks. Profiles of these deployments were built using the three algorithms (STIDE, BoSC and Hidden Markov Models (HMM)) as a way to test their performance using the testing traces, produced through the workload and the injection of attacks, and results were analysed through metric calculation.

The results of the experimental campaign *indicate that the algorithms are applicable in this domain*. In fact, several configurations obtained very good results in all scenarios: in Docker setups (recall `>=0.98` and precision `>=0.72`) and slightly worse, but yet good in LXC setups (recall `>=0.88` and precision `>=0.70`). Finally, the volumes of data generated by LXC prevented the use of the HMM algorithm in its traces and even for Docker it was necessary to use much smaller sizes of training workloads (30 minutes, 1 hour and 2 hours) leading to worse results in most cases (recall `>=0.79` and precision `>=0.66`). It is worth to note that the mentioned results were selected as representative of probable scenarios (see Section 5.2.5) and that the complete results are discussed throughout Section 5.2.

The experimental campaign included a set of experiments using a virtual machine as setup, instead of containers. The idea was to understand what would be the difference in the results, beyond the practical differences in terms of defining the monitoring surface of the system and in terms of flexibility of deployment, replication and failover. The observed results for the OS setup were worse than for Docker and LXC (recall `>=0.56` and precision `>=0.89`). A more complete study with other configurations would be necessary to draw definitive conclusions, but the results indicate that the difficulty in defining the monitoring surface can lead to sub-optimal results.

1.1 Contributions

The main contributions of this work are summarised below:

- **A preliminary analysis focused on the definition of profiles by different anomaly detection algorithms** (presented in Section 4). This analysis studies the impact of different parameters on reaching a learning steady-state for profiles, moment from which we consider the profile as stable enough in order to stop the training procedure. Therefore, this analysis allowed to understand the best parameters and configurations to plan and obtain proper methodology instances.

- **Proposal of an experimental methodology to evaluate the effectiveness and applicability of intrusion detection algorithms to containerised systems** (presented in Section 5.1). The proposed methodology allows to study the applicability and effectiveness based on representative and meaningful metrics which can be extended in order to fit the use case scenario of each situation. In addition, it also allows the adaptation to other container infrastructures.

- **The creation of datasets containing data collected from containers operating under representative workloads** (presented in Section 4.4). These datasets provide the basis to train the classifiers and test them and other tools, such as fully-working IDSes, which aim to work in container environments. All the traces, data and results are available online and can be used for validation and for future works:
  – `https://github.com/jeflora/containers-ids-evaluation`

- **An experimental evaluation of three intrusion detectoin algorithms, in containerised systems deployed into Docker and LXC technologies** (presented in Section 5.2). In general, the algorithms under test were able to produce better results for Docker containers than for LXC. The experimental evaluation allowed to conclude that for Docker containers the classifiers produced higher values for recall and precision than for LXC while also producing a slightly higher false positives rate.

- **A comparison with traditional OS-level deployments as a way to study possible differences between both technologies and deployment modes** (presented in Section 5.2). The results showed classifiers produced higher values of recall for containerised systems (LXC and Docker) while for precision and false positive rate the testing for the traditional OS-level deployment generated more satisfying results. These observations are motivated by the difficulty in defining the monitoring surface for the OS-level deployment of the DBMS under monitoring.

1.2 Document Structure

The remaining document consists of the following five chapters.

Chapter 2 presents the key concepts necessary to understand this work, explaining the concept of intrusion detection and the application of intrusion detection systems, as well as the methodologies for their evaluation. Continuing on to approach the background

on containers and monitoring tools, and reviewing some work conducted in the field of intrusion detection.

Chapter 3 details the objectives of this work, delimits its scope in terms of the relevant threats to be addressed and introduces the approach followed during the duration of the dissertation work.

Chapter 4 focuses on the preliminary experiments conducted before the main experimental campaign, regarding the data monitoring tools elicited and tested to select the most appropriate for this campaign. Moreover we also present the preliminary experiment focusing on the definition of stable containers profile, and closing with an overview analysis of the datasets produced.

Chapter 5 consists of the main contribution of this work, detailing the preparation of the campaign and the setup utilised. It also sheds light on the followed procedure, and its characteristics such as workload features, metrics used to assess performance and finally presents the results obtained from different perspectives and their discussion.

Chapter 6 provides closing remarks and summarises the meaning of this work as well as the future work which can be conducted on top of it.

This page is intentionally left blank.

Chapter 2

Background and Related Work

A large body of research has been developed in the last decades in terms of intrusion detection algorithms, techniques and tools [22]. This includes both commercial and non-commercial Intrusion Detection Systems (IDSes), which have been developed for many domains, environments and systems. IDSes are frequently deployed in cloud deployments [12], not only network-level, but also host-level, as many research efforts have been applied in intrusion detection for virtual machines and hypervisors.

The contributions in terms of container-level approaches are sparse and with limited ability to generalise. In this chapter we cover the key intrusion detection notions and algorithms usually used within this field (see Section 2.1), as well as IDSes and methodologies for the evaluation of these systems (see Section 2.2).

We also describe the key concepts regarding containers' technology and their monitoring (see Section 2.3). Finally, we discuss relevant related work in terms of security in container-based systems (see Section 2.4), including the few existing intrusion prevention and detection systems for the domain (see Section 2.4.1) and also the works of security analysis, which provide interesting information in terms of the most relevant and dangerous vulnerabilities (see Section 2.4.2).

2.1 Intrusion Detection

Intrusion Detection has been applied as a security measure for more than two decades [22]. This technique is the process through which events taking place in a computer system or in a network are monitored and analysed with the main goal of detecting signs of intrusions [1]. Such term is defined as an attempt to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network [1]. Intrusions are caused by malicious agents, who have the objective of obtaining unauthorised access to systems, or authorised systems users, who attempt to gain higher privileges for which they are not authorised.

2.1.1 Intrusion Detection Approaches

Generally, there are three main approaches to detect attacks to a system or network: misuse detection, anomaly detection and hybrid detection.

Misuse Detection

This approach consists of a compilation of previously used attack patterns, and utilises them to identify their occurrence on network traffic or computer system. These attacks, typically, exploit known vulnerabilities with attacks that have been already used in past actions.

This approach is very effective in detecting attacks without generating a high number of false alarms, however, it lacks the possibility to improve its detection span without constant updates to the signature database. Moreover, the misuse approach permits to quickly and reliably diagnose the attack tool or technique being utilised, nevertheless, the inability to adapt and detect variations of known attacks is a great disadvantage.

An example of misuse detection application in practice is the *Snort IDS*[1].

Anomaly Detection

This technique relies upon the assumption that illegitimate activity, which may take place in a system or network, differs, in some manner, from the behaviour which characterises the normal activity of it. Thus, the creation of a profile of a system or network, during legitimate activity, using data, such as the normal behaviour of users or network connections, allows to detect deviations, therefore, detecting the occurrence of intrusion attempts [23].

These features allow to detect new attacks, without requiring previous knowledge about them, which can in turn be used to define signatures for misuse detection. Nonetheless, this approach commonly produces numerous false alarms due to the high difficulty of defining a stable profile of a system or a network, which requires a broad set of data in order to produce reliable results.

This intrusion detection approach has been used widely and some examples of its usage are [24, 25].

Hybrid Detection

The hybrid detection approach uses both misuse detection and anomaly detection methods [11]. Thus, this method allows to detect both known and unknown attacks.

An example of usage for this detection approach is the work of Modi and Patel [26].

[1] `https://www.snort.org`

2.1.2 Intrusion Detection Systems

An IDS can be referred as a piece software which permits the automation of the intrusion detection process [11]. Additionally, there are different methods by which IDSes can be designed, as distributed or not, there are also different monitoring methods, real-time or polling and the target for which is implemented, host or network.

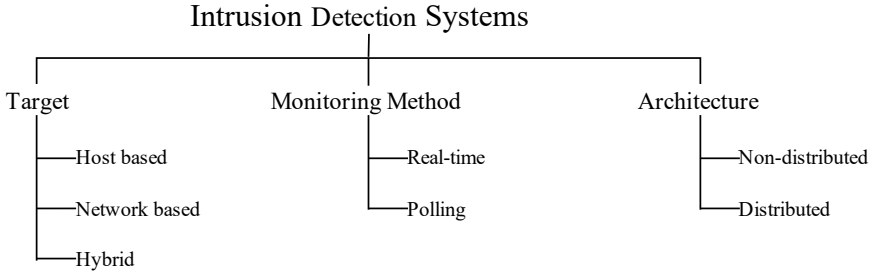


Figure 2.1: Intrusion Detection Systems classification (adapted from Fig.2.3 of [11])

.

Monitoring Target

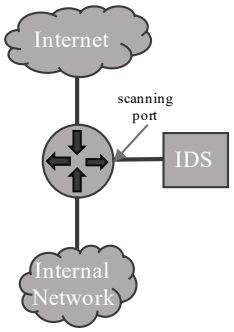
An IDS has a main monitoring target and keeps it under close observation, collecting data about it and trying to detect intrusion attempts against it. There are two main types of targets, which are either machines, that is host based, or network based, however, in some occasions, there is the possibility to monitor both types of targets.

A **Host-based Intrusion Detection System (HIDS)** monitors the characteristics of a host and the events happening within it in order to detect malicious activity. This aspect allows HIDSes to analyse activities with high reliability and precision [23], being possible to conclude with exactitude the origin of an attack against the Operating System (OS), namely which users have been involved and which processes were affected. Typically, these mechanisms utilise OS events, such as system calls or inter-process communication, and system logs produced during its operation, monitoring them and trying to detect the existence of malicious actions within them.

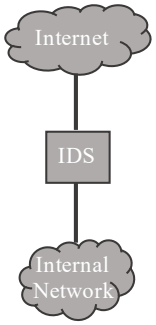
These characteristics concede HIDSes the ability to detect code execution attacks by monitoring activities, such as stack and heap usage or system calls issued by a process and file system violations, through the use of mechanisms as file integrity checking or unauthorised file access attempts. The main reasons for such results are the ability to monitor local events and, when the network traffic is encrypted, HIDSes are not affected because the information is decrypted when it arrives to the destination host. However, these IDSes are costly to manage due to its well-defined and narrow configuration, working only with that precise machine, and also occupy storage on the host they are in as well as computing resources, which may result in performance degradation for the system under monitoring [23].

Among the IDSes which apply host-based intrusion detection are the *OSSEC IDS*[2] and

[2] `https://www.ossec.net`



(a) Network-based IDS passive mode deployment.



(b) Network-based IDS inline mode deployment.

Figure 2.2: Network-based IDS deployment modes.

the *Sagan IDS*[3].

A **network-based IDS** monitors network traffic between devices present in network segments, analysing the packets of network stack's layers, such as the transport layer and the application layer, in order to locate malicious actions. Typically, this type of IDSes is often deployed in dedicated machines or sensors, which permits to more easily ensure its security due to the single-task devices are responsible for. These devices are generally placed as a passive asset on a network, observing a copy of the traffic packet's data, such as origin and destination of the packet, and raising alerts, if applicable. However, on some occasions, IDSes are also placed as any other device present on the network, this is denominated *inline mode*, and these analyse actual traffic passing through them. Figure 2.2 depicts these two modes of deployment.

As a result of its placement, IDSes can gather a huge amount of information regarding the network in which they are deployed, such as hosts identification, applications communicating and even the OSes running on the machines. So, all this information can be processed and logged for forensics procedures to happen when required. Moreover, these security measures are able to detect application layer, transport layer and network layer reconnaissance and attacks, as for instance password guessing, port scanning and IP spoofing [27]. Nonetheless, they cannot analyse encrypted traffic or process fragmented packets and also face complications to handle high number of packets, for example when a network is very busy, which may result in undetected attacks [23].

An example of an IDS which monitors network traffic and detects intrusion attempts is the *Suricata IDS*[4].

Hybrid IDSes are a combination of the two types of IDSes described above. Typically, hybrid IDSes are deployed on the host that aims to monitor, as a host-based IDS, [11] and uses its network connected interfaces to monitor the ongoing traffic. These systems try to combine the advantages of both network and host-based IDSes.

For instance, *Splunk*[5] is a system that is able to monitor both network and hosts.

[3] `https://github.com/beave/sagan`
[4] `https://suricata-ids.org`
[5] `https://www.splunk.com`

In this work, we are focusing on HIDSes as a manner to closely and efficiently monitor containers running in a multi-tenant environment on a host.

Monitoring Method

Another characteristic which distinguishes IDSes is the monitoring method applied by it. Such method can either be real-time, monitoring the events as they happen, or *polling*, not intercepting the events when they take place but rather receiving them asynchronously.

IDSes, which monitor events in **real-time**, analyse system and/or network activities as they occur [11]. The main advantage of these IDSes is the capability to detect attacks as they happen but, unfortunately, this results in the addition of some overhead which may result in performance loss for the system under monitoring. However, the disadvantage may be compensated by the fact that counter-measures against the attack can still allow to minimise its damage or even prevent it.

An IDS that monitors events in real-time is, for instance, the *Snort IDS*[6].

On the other hand, ***polling* IDSes** do not intercept ongoing actions with the objective to obtain information to analyse, instead these devices obtain the required data in other ways, such as an asynchronous retrieve of system logs or periodically observations of the monitored system [11]. These systems do not, therefore, introduce any overhead in the normal flow of information between components of the system under monitoring, but, inevitably, they cannot produce an alert as soon as the attack takes place, which means that the damage caused by it can be substantially larger than what it could be, if acted upon it when first detected.

An IDS that monitors events asynchronously is the *OSSEC IDS*[7].

Architecture

The architecture of IDSes is also a distinguishing characteristic. An IDS can be either distributed or non-distributed, being composed of multiple nodes in the first case or by a centralised component in the last case.

Non-distributed IDSes, also called centralised IDSes, consist of the traditional manner of an IDS deployment, in other words, the system is deployed locally, either at network or host level, and performs the intended functions [11]. These devices do not have a global notion of the machines present in the network segment neither about the network itself, which might make them less successful in detecting coordinated attacks to a set of hosts.

Any traditional IDS is a good example of a non-distributed IDS, such as *Snort IDS*[8].

Distributed IDSes consist of multiple intrusion detection sub-systems, denominated nodes, which exchange relevant data to the detection of possible intrusions. There are two communication possibilities, the nodes can establish communication links between

[6] `https://www.snort.org`
[7] `https://www.ossec.net`
[8] `https://www.snort.org`

themselves, such as a peer-to-peer network, or with a centralised node, which aggregates the data collected by the remaining nodes in order to manage them or conduct further analysis on the information available [11]. This type of architecture permits to have a global notion of a network's or hosts group state, which fills the gap left by the non-distributed IDSes through sharing information of what is happening in multiple sites.

One of the first distributed IDSes is DIDS [15], which was introduced in 1991.

2.1.3 Algorithms for Intrusion Detection

In this section, the focus relies upon algorithms used for intrusion detection, both for misuse and anomaly detection. A description and previous uses are following provided.

Algorithms for Misuse Detection

Typically, the misuse based approach to intrusion detection utilises rule-based algorithms. Such algorithms define the patterns of known attacks and then compare them to new events, aiming to identify intrusion attempts.

Despite the main IDSes available on industry are misuse based, on this work we are focusing on the anomaly detection approach, given its adaptive potential through profile definition and analysis of deviations from it.

Algorithms for Anomaly detection

This sub-section approaches with some detail the algorithms most used for anomaly detection. In addition, it also provides some examples of previous work of their use in intrusion detection.

Artificial Neural Networks (ANNs) consist of a graph-like structure with multiple units of processing (nodes) and weighted connections between them. Each connection's weight determines the influence of a node's value in the outcome of the output node [28]. ANNs' nodes are divided into three subsets: the input, the middle and the output nodes. The input nodes are the starting point of the computations which are spread to the middle processing units through existing connections, finally arriving at the output nodes, where final results are produced [28].

The **Bags of System Calls (BoSC)** method is based upon a window sliding over a trace of system calls utilised to detect intrusions and to define a baseline behaviour database, which contains *bags* of system calls considered normal. *Bags* do not take into account the order by which the system calls are issued but rather focus on the frequency of each system call within the window being processed [29].

Hidden Markov Models (HMM) is structured as a set of states, a matrix of emission probabilities, corresponding to the probability of observing a symbol given the current state, and a matrix of transition probabilities, which are the probability of transitioning from a state to another given an observation. This method has also been widely used in

intrusion detection, where a sequence of events is considered anomalous if the probability of it belonging to the defined profile is below a certain, user-defined, threshold [30].

K-Nearest Neighbour (KNN) is a Machine Learning (ML) classifier, which after a training phase, defines a model able to identify anomalous events. This technique is based on distance heuristics between new data and classified data, as a manner to group the events into different classes according to event similarity. Specifically, when a new event is processed the K nearest neighbours to it are calculated and the most frequent label among them is assigned to the new event [31].

The **Naïve Bayes (NB)** method is a statistical approach based on the Bayes theorem and on the assumption that every feature is completely independent. The NB method is a widely used classifier due to its high scalability and effectiveness. Therefore, this method allows classifying new events based on previous ones, resulting in the ability to identify possible intrusions with a given probability [32].

The **Sequence Time-Delaying Embedding (STIDE)** method is based upon a window sliding over a trace of system calls utilised to detect intrusions and to define a baseline behaviour database, which contains the sequences considered normal. This technique was proposed in 1996 [13] and is still in use today, it maintains the original order of system calls, contrary to BoSC which is frequency based.

Due to the relevance of BoSC and STIDE in our work, it is of utmost importance to comprehend them fully, therefore, Figure 2.3 depicts an example of their usage, similarities and differences. The image comprehends the step-by-step windows and final normal behaviour database produced by each algorithm.

Support Vector Machines (SVM) is a classifier method, which can be used to detect anomalies. On its essence this method uses prior knowledge, acquired upon training, to divide events into different classes separated by a hyper-plane, which is defined by a number of support vectors used to define boundaries [33].

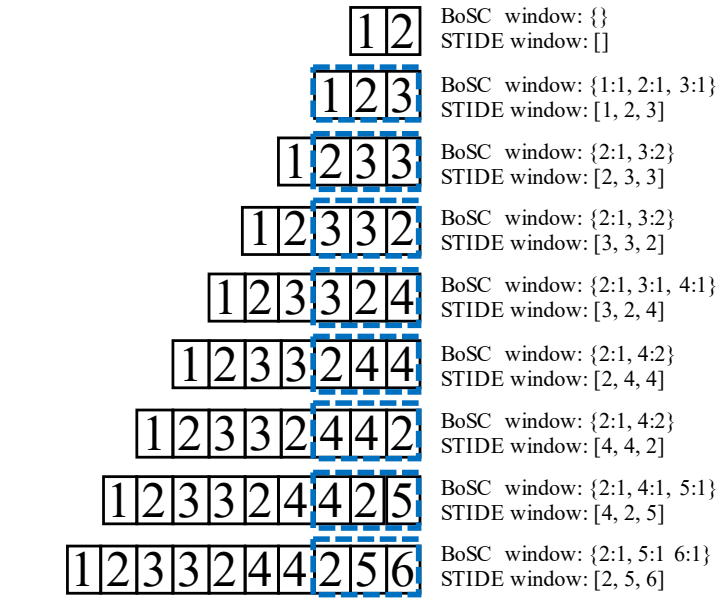
In anomaly detection, it is usual that SVM is used with only one class, being known as One-Class Support Vector Machines (OCSVM), where this class defines the normal behaviour. **OCSVM** removes the need to provide anomalous traffic to the model during training. This characteristic also increases the probability of detecting unknown attacks since it is not trying to classify them [34].

2.1.4 Summary

In this section, we have approached intrusion detection, covering the approaches that can be followed to detect intrusion attempts, these are exposed on Table 2.1.

Furthermore, we have also studied IDSes and their main characteristics are exhibited on Table 2.2.

The anomaly detection algorithms were analysed and a systematisation of advantages and disadvantages is presented on Table 2.3.



Profiled BoSC Behaviour DB	**Profiled STIDE Behaviour DB**
{1:1, 2:1, 3:1} ; {2:1, 3:2} ; {2:1, 3:1, 4:1} ; {2:1, 4:2} ; {2:1, 4:1, 5:1} ; {2:1, 5:1 6:1}	[1, 2, 3] ; [2, 3, 3] ; [3, 3, 2] ; [3, 2, 4] ; [2, 4, 4]; [4, 4, 2] ; [4, 2, 5] ; [2, 5, 6]

Figure 2.3: Exemplification of BoSC and STIDE operation.

2.2 Evaluation of Intrusion Detection Systems

In this section, we present the state of the art regarding evaluation of IDSes. This procedure is very important in order to understand the performance of a system in detecting intrusion attempts, thus this section approaches the generation of workloads used to test IDSes, and metrics and methodologies utilised to assess their performance.

2.2.1 Workloads

In order to evaluate an IDS, it is required to utilise workloads, so that the system has information to process and, therefore, produce results which can be analysed, Figure 2.4 depicts the structure of workloads' definition. These workloads can be either benign, which consist of *normal*, non-malicious interactions, or malicious, which consist of the former's opposite. Moreover, these workloads can be used separately or combined, thus resulting in three content types for workloads: **pure benign**, which consist of only non-malicious activity; **pure malicious**, constituted of attacks only; and **mixed workloads**, which contain both types of data, malicious and non-malicious, and are typically used for realistic testing scenarios [18].

In addition, workloads can also be categorised according to their form, which can be one of executable form or trace form. While the **executable form** is usually used for live testing an IDS, thus evaluating events as they happen, the **trace form** consists of live execution recording, thereby allowing the repetition of the experiment multiple times with the same

Table 2.1: Summary of Intrusion Detection Approaches.

Intrusion Detection
Intrusion Detection Approaches (Section 2.1.1)
Misuse Detection
• This approach is based on prior knowledge about attacks • Low false positive rate • Unable to detect unknown attacks, or variations of known attacks
Anomaly Detection
• Higher false positive rate • Consists of two phases: a training phase and a detection phase • Can detect *zero day* attacks
Hybrid Approach
• Uses known patterns to detect attacks and profiles to detect *zero day* • Efficiency is lower when compared to single-technique approaches

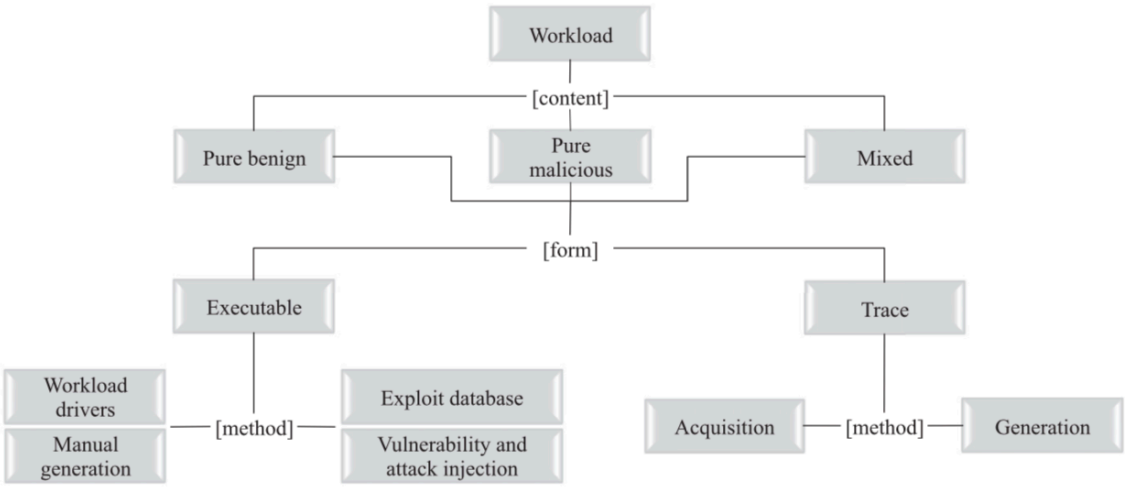


Figure 2.4: IDSes evaluation workload types (from [18]).

data. Generally, the trace form is more often utilised due to the fact that executable form requires an explicit victim environment which can be very expensive to establish [18].

There are two methods to generate pure benign executable workloads, the use of workload drivers or their manual generation. Regarding **workload drivers**, these are typically utilised to generate pure benign activities, mainly aiming to test the system with different types of interactions, however, these drivers do not correctly recreate the normal interactions with a system [18]. An example of such workload drivers is the *mysqlslap* [9] tool, which allows to perform queries to a MySQL database and permits the user to select multiple configurations, such as number of requests and clients interacting with the system.

With respect to **manual generation**, which might be an acceptable alternative to workload drivers, it consists of the execution of real tasks by real systems and/or users. This method allows to overcome the issue faced by workload drivers, that do not generate real-

[9] `https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html`

Table 2.2: Summary of Intrusion Detection Systems.

Intrusion Detection Systems
Monitoring Target(Section 2.1.2)
Host based
• Monitors a machine locally • Analyses information of that machine, such as logs or resource usage
Network based
• Deployed on a dedicated machine, monitoring the ongoing traffic on a network • Can be deployed either *inline* or in *passive* mode
Hybrid
• Can monitor both targets • Typically deployed at the host target, and uses its network devices to monitor the network
Monitoring Method (Section 2.1.2)
Real-time
• Monitors events as they happen • Adds overhead to the normal execution of the target
Polling
• Obtain the data to analyse in an asynchronous manner • Do not raise alerts when intrusions happen
Architecture (Section 2.1.2)
Non-distributed
• Traditional IDSes, consist of a centralised node • Do not possess a global notion of what is occurring on the network segment or machines being monitored
Distributed
• Can detect orchestrated attacks • Consist of multiple nodes, which share information, normally through a central node

istic activities, since manual generation, if based on a realistic activity model, generates data similar to what a system in normal operation would do [18]. Nonetheless, with this approach it is not possible to customise the workload in terms of, for example, activity intensity.

Changing the focus to pure malicious workload, there are also two approaches, which can be used to produce them, those are exploit databases and vulnerability and attack injection [18]. For **exploit databases**, they can be viewed as penetration testing tools, such as *Metasploit* [10] or *Nessus* [11], that have a database for itself, which allows exploiting known vulnerabilities of certain systems. Moreover, there are also multiple exploit repositories available online, which can be used to gather working proof of concept code that successfully exploits a known vulnerability.

Focusing on **vulnerability and attack injection**, a technique based on fault injection,

[10] `https://www.metasploit.com`

[11] `https://www.tenable.com/products/nessus/nessus-professional`

Table 2.3: Summary of Anomaly detection algorithms [+:advantage −:disadvantage].

Algorithms for Anomaly Detection
Artificial Neural Network
+ Robust to outliers
+ Learns feature interaction automatically
− Slow training speed
− Computationally expensive
Bags of System Calls
+ Reduces database size because it is frequency based
− The loss of sequence reference as a consequence of being frequency based
Hidden Markov Models
+ Suitable to be used in application that deal with data based on sequence of feature
− Computationally expensive
K-Nearest Neighbour
+ Fast training
− Slow in the predicting phase
− Requires large size of samples
Naïve Bayes
+ Fast training and prediction phase
− Fails estimating rare occurrence
Sequence Time-Delaying Embedding
+ Takes into account sequence of occurrence
− Database size can grow large if window size is high
Support Vector Machines
+ Good theoretical guarantees regarding over fitting
− Can be inefficient to train, memory-intensive
One-Class Support Vector Machines
+ Does not require malicious workload
− Can be inefficient to train, memory-intensive

it grants the possibility to exploit vulnerabilities added to a certain system. This method is often used when it is not possible to use publicly available exploits, either because they do not exist or the vulnerable version of the system under monitoring is not available.

Finally, with regard to trace form generation, there are two methods that allow to generate any of the three types of workloads (pure benign, pure malicious and mixed), those are **trace acquisition** and **trace generation**. Regarding **trace acquisition** there are two possibilities to achieve this. The first is to collect traces from real systems when in production from organisations, however, companies tend to be unwilling when it comes to share real data of their systems due to privacy concerns [18]. Secondly, there are publicly available traces that can be used to train and test the IDSes, however, these traces tend to be considered outdated due to its long time existence [35]. In connection with **trace generation**, there are also two main ways to utilise this technique, that are generating traces in a *testbed* environment and the use of *honeypots* to record interactions with it.

Focusing on the use of a **testbed environment**, these testing systems can be utilised as a target for the attacks performed, in order to collect traces, using the techniques described previously, such as the use of workload drivers and the manual generation method. Still, this technique may generate traces which are not representative of realistic interactions with services, since results collected in small environments do not usually apply to larger ones [36]. As for ***honeypots***, which intend to simulate the operation of real systems, it is possible to collect the interaction of attackers who interact with them, providing the traces which are required to evaluate IDSes. Since the actors involved are not aware to the fact they are interacting with a fake system, this approach permits to generate realistic traces, despite being harder to classify the traces produced, for instance in terms of attack types.

2.2.2 Metrics

So that it is possible to understand whether an IDS is performing accordingly to what is excepted, there is a need to compute meaningful metrics values. These metrics allow to understand, from various perspectives the results obtained, such metrics can be related to performance and security. This Section aims to outline and explain the metrics usually utilised when carrying out an IDS evaluation procedure.

Regarding performance metrics, which are not the focus of this work, it refers to the non-functional properties, such as capacity and resource consumption [37]. The focus of this work is related to security metrics based on the results produced by an IDS and the ground truth knowledge about the workload being utilised. Although not all IDSes output results in a binary form, either normal or anomalous, the ones which produce a different output, such as a probability or a multi-class classification (different from either anomalous or normal) are easily converted to this output form. The basis for IDSes evaluation is, therefore, the comparison of the expected results against the obtained results, which produce four possible combinations as shown in the confusion matrix depicted in Table 2.4.

Table 2.4: Confusion matrix of IDSes results categorisation.

Expected	**Obtained**	
	Normal	Malicious
Normal	**True Negative (TN)**	**False Positive (FP)**
Malicious	**False Negative (FN)**	**True Positive (TP)**

The classifications produced by an IDS are categorised into one of the four categories presented in Table 2.4. When the event baseline classification is **normal**, that is non-malicious, and the IDS considers it to be normal then we achieve a **true negative** category because the IDS classified the event correctly and it was not an intrusion attempt, but if the IDS had categorised it as **malicious**, then it would be categorised as **false positive** instead due to a misclassification of a non-malicious event. On the other hand, if an event's baseline classification is **malicious** then the result produced by the IDS is considered as a **false negative** when the IDS does not raise an alert and as **true positive** when it does. Therefore, based on these four categories, it is possible to define several metrics, as defined in [38].

Firstly, the proportion of malicious cases classified as malicious denotes the metric **Recall**,

also known as true positive rate or sensitivity. Although this metric is only focused on the anomalies detected, it provides a good understanding on IDSes performance regarding the detection of the true anomalies.

$$recall = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{2.1}$$

In addition, **Precision** consists in the ratio between correctly classified malicious events and all the events classified as malicious by the IDS. However, this metric, as well as recall, is vulnerable to bias by *prevalence* and *skew* [39].

$$precision = \frac{TP}{TP + FP} \tag{2.2}$$

Moreover, **F-Measure**, or $F_1 Score$, is built upon precision and recall, being the harmonic mean of both metrics. Although this metric assigns the same cost to true positives and false positives, it provides a good understanding on which IDS detects high number of anomalies while reporting a low number of false positives [38].

$$f - measure = 2 \times \frac{recall \times precision}{recall + precision} = \frac{2 \times TP}{2 \times TP + FN + FP} \tag{2.3}$$

Finally, **False Positive Rate**, or miss rate, is the proportion of normal events classified as anomalous [38].

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{2.4}$$

Furthermore, another very utilised representation for IDS selection is a Receiver Operating Characteristic (ROC) curve which plots the true positive rate (recall) against the false positive rate of an IDS with varying configurations, so that it is possible to compare them and identify the one that fits best.

A ROC curve plot example is available on Figure 2.5.

However, some argue that ROC curves do not take into account all the meaningful and relevant data to assess the performance of an IDS [41]. Consequently, as a manner to overcome such limitations, the expected cost analysis was proposed in 2001 [41]. This approach takes into account not only the ROC but also on cost metrics and the hostility of the operating environment through the probability of intrusion [41]. Gaffney and Ulvila, define the expected cost based on the probability of intrusion, thus taking into account the hostility of the environment, the false positive rate and recall values, and the cost proportion associated with responding to a non-intrusion and not responding to an intrusion.

Thus, the expected cost can be calculated through the following equation:

$$ExpectedCost = Min\{C\beta p, (1-\alpha)(1-p)\} + Min\{C(1-\beta)p, \alpha(1-p)\} \tag{2.5}$$

Having: $C = \frac{C_\beta}{C\alpha}$, $\alpha = FPR$, $(1-\beta) = Recall$, $p = P(I) = \frac{TP+FN}{TP+FN+TN+FP}$

In our practical work, we are using $C = 10$, based on previous work [41], which means that it is ten times more costly to fail to respond to an intrusion than it is to respond to a non-intrusion. In addition, the formula to calculate the probability of intrusion was defined taking into account our practical scenario.

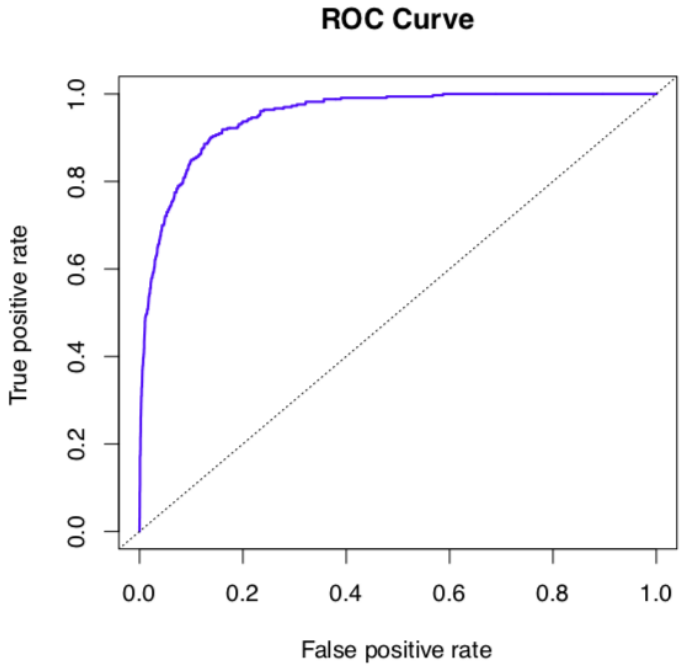


Figure 2.5: ROC curve plot example (from [40]).

2.2.3 Summary

In this section, we have exposed the background related to IDSes evaluation, such as workload types and metrics used. There are mainly three types of workloads, **pure benign**, **pure malicious** and **a mix of both**. Regarding metrics, we have five different and each has a purpose.

In this summary, we systematise the advantages and disadvantages of workload generation manners (Table 2.5) and metrics used to measure IDSes performance (Table 2.6).

2.3 Containers

The basis for containers was introduced on the late 1970's, more precisely on 1979, when the *chroot* [42] system call was introduced into UNIX. This system call enables the modification of a process's and its sub-processes' execution root directory. As a result, it became possible to isolate processes' execution, one of the main characteristics of containers - **isolation**. Such **isolation** turned attainable through file-system segregation, partitioning it into multiple branches and assigning each one to a different process thereby assuring processes' inability to access files above their assigned sub-tree. Nevertheless, this was still the beginning of process isolation.

Later on, in 2000, the FreeBSD Jails [12] was released, almost twenty years after *chroot* addition to BSD, and was a big step towards present containers. As a manner to achieve a complete separation between systems, Jails allowed to partition the BSD system into multiple independent sub-systems, consisting of a virtualisation of the filesystem, the users

[12] `https://www.freebsd.org/doc/handbook/jails.html`

Table 2.5: Workloads summary [+:advantage −:disadvantage].

Workloads
Executable Form
Workload Drivers
+ Allow customisation of the interactions produced
− Do not correctly emulate real interactions with systems
Manual Generation
+ May emulate real systems properly when based on realistic activity models
− Supports no customisation of the workload produced
Exploit Database
+ Ready to use exploits
+ Generates realistic workloads
− Some exploits only work for specific versions of the target which may not be available anymore
Vulnerability and Attack Injection
+ Permits to inject vulnerabilities on any software version
− Slow process since it requires source code manipulation
Trace Form
Trace acquisition
+ Real world and large size workloads
− Companies are not willing to share traces from production environments due to privacy issues
− Traces available online may be outdated
Trace generation
+ A *testbed* allows an easy data classification
+ The use of *honeypots* may permit to collect real interactions with systems
− The use of *testbed* environments may result in non-representative workloads
− It is hard to classify traces collected from *honeypots*

set and the networking subsystem resulting in the possibility to assign a dedicated IP address [43]. Along with BSD Jails was also introduced the notion of **operating system-level virtualisation**.

In the following years, other technologies similar to FreeBSD Jails were released, namely *Linux VServer* [13] on 2001, *Solaris Containers* [14] on 2004 and *OpenVZ* [15] on 2005. However, it was not until 2006 that another great step was taken, with the release of Process Containers by Google. This was designed to be used for limiting, accounting and isolating resource usage (CPU, memory disk I/O, network) of a group of processes. Later, it was renamed to *control groups (cgroups)* and added to the Linux kernel on 2008. In the same year, LinuX Containers (LXC) were released. Such technology is built upon *cgroups* and *namespaces*, features present in the Linux kernel. While *namespaces* provide process isola-

[13] `http://linux-vserver.org/Welcome_to_Linux-VServer.org`
[14] `https://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html`
[15] `https://openvz.org`

Table 2.6: Metrics summary [+:advantage −:disadvantage].

Metrics
+ Combined are good evaluation mechanisms
+ ROC curves are a great manner to identify the best configuration for IDSes, despite some limitations
+ The expected cost calculation allows to take into account the hostility of the operation environment
− Require ground truth to be calculated
− Some metrics, if used isolated, can be misleading

tion into groups, *cgroups* is responsible for controlling the resource usage for each container within the main system.

Another important step in containers' evolution was the release of Docker [16], on 2013, which really pushed forward the adoption of this technology. Docker really eased application development and deployment which resulted on massive adoption of containers by cloud service providers. Service providers, such as Amazon with Amazon Web Services (AWS) or Microsoft with Azure, have already put forward massive investment in this technology.

Figure 2.6 depicts the evolution described.

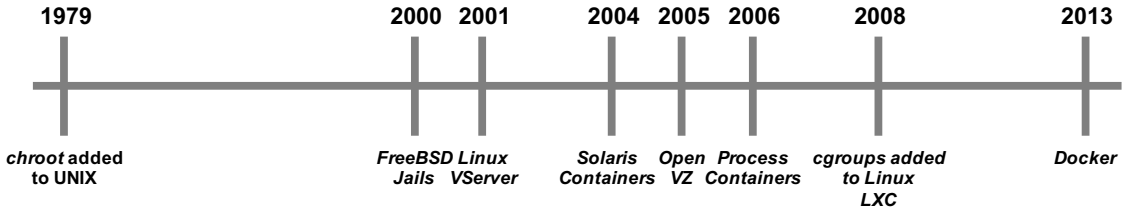


Figure 2.6: Containers evolution timeline.

So, this technology is built upon Linux's *namespaces* and *cgroups*, which permits to group a set of processes and resources, such as memory or CPU, and to isolate them from processes and resources outside of it. All containers on a host machine share its kernel functionalities, as a result providing a **lightweight** virtualisation mechanism.

There are two types of containers as depicted in Figure 2.7.

System Containers are characterised by the virtualisation of a OS that is lighter and faster to boot than a Virtual Machine (VM). This technology, built upon *cgroups* and *namespaces*, can run multiple services within them as a normal server would. Some examples are the *LXC* and *OpenVZ* technologies.

Application Containers are a step forward of system containers. These are specialised containers, meant to run only one service (application) at a time. Docker is one of the examples of this technology.

[16] `https://www.docker.com`

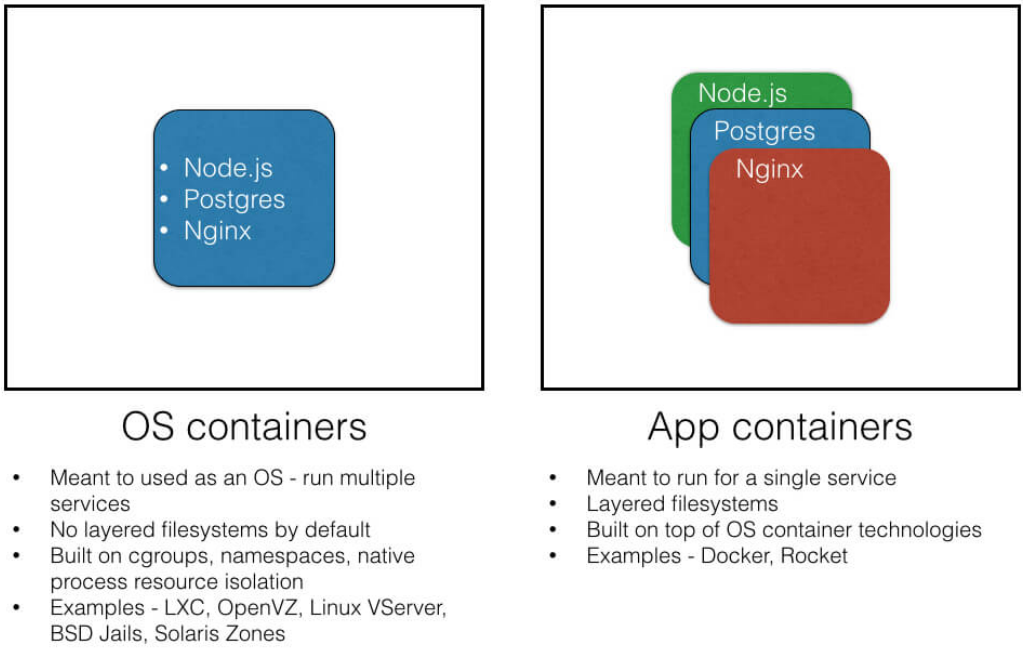


Figure 2.7: Application Containers and System (OS) Containers (from [44]).

2.3.1 Containers Concepts

With respect to concepts, which are provided by containers, the most important are **isolation**, and **resource limitation**. These characteristics are achieved due to the use of two features available in the Linux kernel, **namespaces**[17] and **control groups (cgroups)**[18].

Namespaces create an isolated instance of a global resource, such as user groups or network devices, by enclosing a system resource into an abstraction available to processes within a given *namespace*. This instance is only available to the processes within the *namespace* to which it was created for. Therefore, any change made to it only affects the processes members of the *namespace*. There are, currently, seven different types of *namespaces*, which are analysed on Table 2.7.

Table 2.7: Types of *namespaces* provided by Linux

Namespace Type	Description
Cgroup	This *namespaces* type isolates the ***cgroup* root directory**
IPC	*IPC* namespaces isolate **System V IPC** and **POSIX message queues**
Network	Responsible for isolating **network devices**, **stacks**, **ports**, among others
Mount	Enclose the **mount points**
PID	Limit the **process IDs**
User	Enforce **user and group IDs**
UTS	Allow the isolation of the **hostname and NIS domain name**

Regarding **cgroups**, the other fundamental feature which makes containers possible, these

[17] `http://man7.org/linux/man-pages/man7/namespaces.7.html`
[18] `http://man7.org/linux/man-pages/man7/cgroups.7.html`

allow the organisation of processes into hierarchical groups whose resource usage can be monitored and limited as fit. This feature provides an interface through *cgroupsfs* and globally handles the aggregation of different processes, furthermore the available kernel sub-systems control resources, such as CPU and memory, and assure that processes within a given *cgroup* do not exceed their limits.

In conclusion, the combined use of both of these features permit to **isolate** a group of processes, which result in a **container**.

2.3.2 Containers Technologies

There are multiple solutions to address containers' technology. The aim of this section is to provide some details regarding the most popular and most used containerisation platforms. Following, the platforms such as LXC and Docker are approached and details are provided.

LinuX Containers

LinuX Containers (LXC) were introduced in 2008, this technology's main goal is to provide an OS-level virtualisation so that it is possible to run multiple isolated containers on a machine. Despite some misconceptions, LXC does not provide virtual machines, instead it provides a virtual environment that has its own resources, such as CPU and memory, and the resource control mechanism [45].

In order to provide the service, LXC is based on Kernel *namespaces* and *cgroups*, as described in Section 2.3.1, *Apparmor*[19], which is a Mandatory Access Control (MAC), and *SELinux*, that is also a MAC focused on system calls and profiles, *Kernel capabilities*[20], which provide to assign superuser permissions to users according to their need, and *Seccomp* policies, which permit to filter system calls issued by containers.

Moreover, this technology allows to create two types of containers, either unprivileged or privileged, although the former are considered safer, due to not mapping the *root* user ID to the *root* user ID of the host machine, some Linux distributions require further configuration to allow their creation [45].

Docker

Docker was first released in 2013, initially named *dotCloud* and based on LXC Docker has, however, shifted away from both. Nowadays, this platform is also based on the same technologies as LXC, such as *namespaces* and *cgroups*, but is not an extension of it.

Docker consists of three main components: the Docker Client, the Docker *Daemon* and the Docker Registry. Regarding the **Docker client**, this component is the responsible to interact with the Docker *daemon* given the user issued commands, it permits to control containers instantiation, and management. For the **Docker *daemon***, this component manages all the containers present on the Docker host and all the images present in its

[19] `https://wiki.ubuntu.com/AppArmor`
[20] `https://wiki.archlinux.org/index.php/Capabilities`

cache. A Docker container is built on a hierarchical combination of multiple images, an image acts as a template for the container. Typically, an image's components are defined through the use of a file named *Dockerfile*, which allows the developer to define custom images, by installing libraries required, creating new users and deploying their applications. With respect to the **Docker Registry**, this component is a remote one, and acts as an image repository and distributor because it maintains both base and custom images, allowing their distribution to other hosts, furthermore it also keeps the different versions of an image, permitting to select a specific version.

In sum, these are the main reasons why Docker popularity growth has been immense, their agnostic functioning and the easy and fast deployment of containers have contributed greatly to it.

2.3.3 Containers Monitoring

After containers are deployed, it arises the need to manage them and maintain them under observation, so that is possible to understand whether a problem happens and action is required. As a manner to keep this technology under examination, information about containers, such as CPU or memory use or even system calls, is used to conclude if the system is working correctly.

This Section provides an explanation of some monitoring tools which can be adopted in this context.

Sysdig

This tool [46] instruments machines at the OS level, installing itself into the Linux kernel and capturing system calls and other OS events. It also provides a command line interface to easily filter and decode such events so that is possible to extract useful information. *Sysdig* can be used in both real-time or in after the fact inspections, through generation of trace files.

The main features provided by *sysdig* are its native support for all Linux container technologies, such as Docker and LXC, and providing a *"unified, coherent, and granular visibility into the storage, processing, network, and memory subsystems"* [47]. Moreover, it makes possible system activity trace files creation, thus enabling after the fact analysis. These files can include the state of the system thereby assuring that the context in which the capture was performed does not get lost, which is available due to the filtering procedures that can be used to process the information collected easily.

Sysdig is a multi-component tool which collects the system events by using a probe, called *sydig-probe* that is responsible for collecting the events, at kernel level, and stores them into a shared buffer for later processing by the other components which work at user level. Following the collection of events, there are two components, *libscap* and *libsinsp*, which read, decode and parse them, by applying for instance high-level filters. Furthermore, acting as a wrapper for all these components is the *sysdig* component, which is the command line tool that receives the arguments and performs parsing and management. Figure 2.8 depicts *sysdig*'s components and their responsibilities.

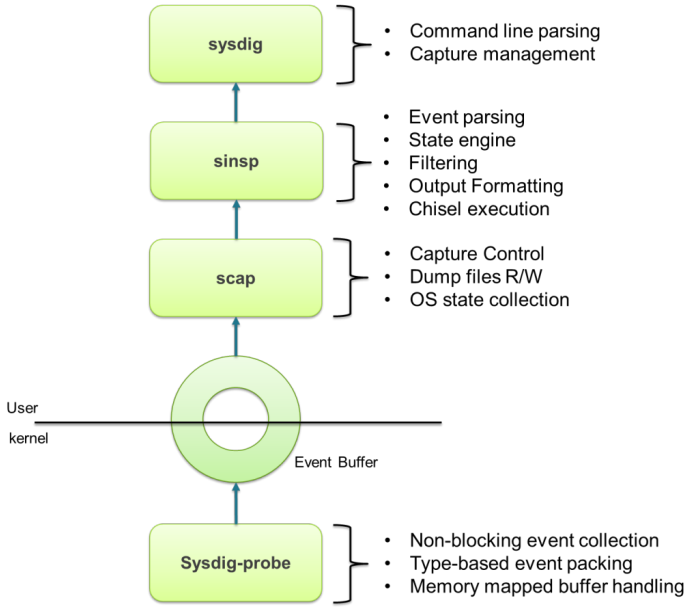


Figure 2.8: *Sysdig* components (from [48]).

strace

This tool [49] consists of a diagnostic, debugging and instructional user space utility managed through a command-line interface for Linux and it can be used to monitor processes and the Linux kernel, enabling the collection of events such as system calls, signal deliveries, and changes of process state. The operation of *strace* is made possible due to a Linux kernel feature known as *ptrace*.

The main features provided by *strace* are the possibility to attach to an already running process, leading to the ability to start monitoring a process at any given time of its execution, to filter data collected by type of system call, such as *file* or *process*, thus gathering only calls related to file handling and process management, respectively. Additionally, this tool also permits to trace only system calls events which access to a user-defined path, to perform a full hexadecimal and ASCII dump of all the data read from/written to file descriptors and also introduce a system call fault injection.

Despite being mainly process-oriented monitoring tool, *strace* can also be used to monitor containers successfully.

The architecture of *strace* is characterised by the introduction of additional contexts switching operations. When a system call is executed, there occurs a context switch from user to kernel mode. However, when *strace* is monitoring the process executing the call, two additional context switches occur. That is, this utility uses the *ptrace* Kernel feature to interrupt the process every time a system call is executed, collect and decode it and return the process normal execution [48]. Therefore, *strace* is performing a blocking collection of data and compromising the performance of processes being executed. Figure 2.9 depicts the execution flow of a system call when *strace* is monitoring the process.

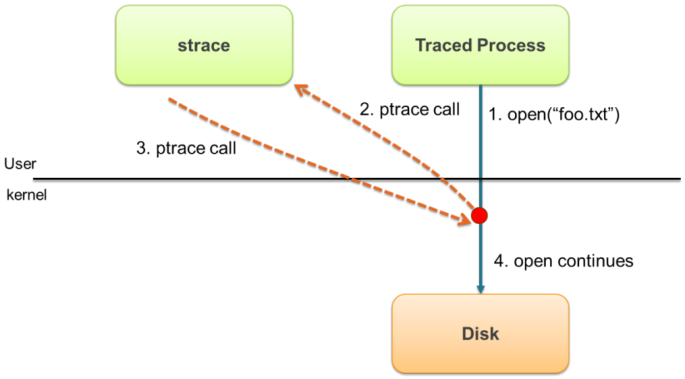


Figure 2.9: *Strace* operation (from [48]).

cAdvisor

Container Advisor (cAdvisor) [50] monitors the resource usage and performance characteristics of running containers. This tool is a *daemon* which collects, aggregates, processes, and exports information, in various formats, about running containers [50]. More precisely, for each container, the tool keeps resource isolation parameters, historical resource usage and network statistics.

As a main feature, it provides a REST API to perform queries upon and therefore retrieve the metrics by it collected. This functionality permits to easily integrate cAdvisor with an information processor, which achieves a specialised task, keeping modularity and independence.

Although cAdvisor provides native support for Docker containers [50], its focus is on performance issues such as CPU usage and advice on containers performance [50].

2.3.4 Summary

In this section, we have focused on **containers**, their history, types and main concepts. Containers can be of two types, either **system containers**, which virtualise the complete OS, or **application containers** that are specialised containers, meant to run only one application. Containers are a set of processes that run in an isolated manner with resource and file system segregation based upon *cgroups* and *namespaces* kernel features.

In addition, some containers technologies, such as LXC, which virtualises system containers, and Docker that is focused on application containers, are also described and studied. Moreover, the tools to monitor containers were also studied, namely *sysdig*, *strace* and *cAdvisor*. These tools characteristics are highlighted on Table 2.8.

Additionally, we approached the works performed regarding intrusion detection at container-level, and to the best of our knowledge, there is the work of Abed *et al.* [16] and Srinivasan *et al.* [17]. However, these works focus their attention on the application running within the container, monitoring only that application, whereas our intentions are to monitor the container as a whole, monitoring it from outside instead of collecting data from the inside and exporting it through mounted volumes into the outside.

Table 2.8: Monitoring Tools comparison.

Tool	Main Focus	Collects System Calls	Collects Resource Usage	Data Collection Manner	Native Support
sysdig	Container-oriented	✔	✔	Passive	OS, Docker, LXC
strace	Process-oriented	✔		Active	OS
cAdvisor	Container-oriented		✔	Passive	Docker

2.4 Security in Containers

2.4.1 Intrusion Prevention and Detection in Containers

To the best of our knowledge, there is a limited number of published research works regarding intrusion detection at container level. Below we discuss these works including some that, although may be not intrusion detection, try to provide the same functionality: detect or avoid security attacks.

In 2015, Abed *et. al* [16] proposed an IDS for container level, which used BoSC to represent the normal behaviour database entries. In this work, an anomaly detection based IDS was proposed, capable of monitoring a container and detect possible intrusion attempts at real time. During the experimental analysis, the authors claimed to have obtained a TPR of 100% with a FPR of 2%. However, the use of *mysqlslap* as a benign workload generator is not a representative manner to emulate real interactions with a Database Management System (DBMS) and therefore the results produced may not be representative as well.

In 2015, Mattetti *et al.* [51] contributed with a framework to protect linux containers and their workloads. For this, they monitored both run and build commands to extract relevant information in order to generate AppArmor profiles for each container as a way to protect the host OS and the container. In addition, the authors referred the insignificant performance impact of their framework.

In 2017, Bila *et al.* [52] propose a quarantining methodology for vulnerable containers identified through a vulnerability scanning service in addition, it isolates the compromised containers from the remaining ones and stores them for further forensic analysis. The authors also propose a policy manager to allows security professionals to introduce policies to enforce compliance and isolation.

In 2019, Srinivasan *et al.* [17] proposed an approach based on the NGram probability. In addition, the authors used Maximum Likelihood Estimatior and Simple Good Turing techniques to obtain probabilities of unseen values. They have also conducted practical experiments, similar to Abed's, using a docker container with a web application within a docker container and the University of New Mexico (UNM) dataset as well. The results showed accuracy values ranging from 87-97%, recall ranging from 78-100% and FPR ranging from 0-14%. Nevertheless, as Abed *et al.*, the authors of this work utilise *sqlmap* to exploit the web application which does not generate a realistic since this exploitation tool is performed through sql injection attacks.

2.4.2 Security Analysis in Containers

Several works have published security analysis of container technologies. These works provide very interesting insight in terms of the most frequent and relevant vulnerabilities, and, in some cases, the reasons behind them. However, these works are not directed to try to solve those issues, and therefore cannot be compared with the present work.

Experiments have shown that there is an increase in the attack surface exposed in the host of the Docker server [53]. The authors used a setup of a Docker server machine (with LXC as the execution driver) and a bare metal server machine. Even when using official repositories, they concluded that the exposure is due to the vulnerabilities exposed by the OS images inside the container.

A survey on Docker security and an analysis of its ecosystem was performed in [54]. The authors analysed the common use cases of docker and concluded that many of the security issues are due to the incorrect selection of containers instead of VMs, as it is not the correct selection in many cases.

The sharing of the host OS kernel have been continuously under study due to high reward in case of its successful compromising. The escalation of privileges achieved through *container escape* attacks can cause huge amounts of damage to the infrastructure [55]. In connection with this concern, Jian *et al.* propose a defence method in order to prevent *contaier escape* attacks in addition to demonstrating it.

In another work, Gao *et al.* [56] reveal information leaking channels in containers, demonstrating that adversaries could explore these vulnerabilities in order to compromise the cloud infrastructure. In addition to demonstrating the existence of information leakage, they propose a power-based namespace as an alternative to mitigate these concerns.

The empirical analysis presented in [57] analysed all the publicly available information on Docker security vulnerabilities and concluded that the "most common causes are unprotected resources and incorrect permissions management" while the "most common consequences are bypass and gain privileges", issues that are paramount in multi-tenant setups. The results also showed that existing static code analysis tools are ineffective in detecting these vulnerabilities.

2.4.3 Summary

In this section, we presented the relevant work conducted with containers security. Our focus remained in work related to container level intrusion detection, nonetheless we have also discussed more general security topics as security assessments of docker vulnerabilities of the proposal of approach to mitigate certain types of attack venues. This provides a wider notion of container security research work conducted and therefore a more concrete basis for our work. Although the majority of work with containers is focused on Docker containerisation platform, some of them could be adapted to work to other containerisation platforms, such as LXC which some works also focus upon.

The main takeaways for each work are compiled, as a item list, and summarised in order to explain the main contributions and proposals of each research work as concise and easier

to understand as possible on Table 2.9.

Table 2.9: Related work summary.

Related work
Abed *et al.* [16]
• Host-based Intrusion Detection • BoSC as a classifier to detect anomalies
Mattetti *et al.* [51]
• Monitors build and run of Docker containers • Builds AppArmor profiles
Bila *et al.* [52]
• Framework for containers protection • Quarantines vulnerable container
Srinivasan *et al.* [17]
• Host-based Intrusion Detection • Apply NGram classifier
Mohallel *et al.* [53]
• Compares security of Linux containers with host base OS • Vulnerability Assessment
Martin *et al.* [54]
• Analysis of the containers security ecosystem • Vulnerability Detection
Jian *et al.* [55]
• Discusses existing security mechanism and security issues of Docker • Defence against escape attacks
Gao *et al.* [56]
• Explore Information leak channels • Propose a two-stage defence approach
Duarte *et al.* [57]
• Docker Security Assessment • Static Code Analysis

Chapter 3

Research Objectives and Approach

This chapter describes the main objectives of this work, outlining our focus and the defined manner to achieve it through an approach designed for the effect. Section 3.1 outlines our goals and provides a description in order to explain them, Section 3.2 highlights the relevant threats for this work whilst Section 3.3 proposes and explains the approach to achieve the defined objectives.

3.1 Research Objectives

The main goal of this work is to **understand the effectiveness and applicability of state-of-the-art host-based anomaly detection algorithms to containers deployed in multi-tenant environments**. For this, it is required to produced datasets of both training and testing data regarding the system calls and events produced by containers deployed in these environments. In addition, we have also to elicit the state-of-the-art algorithms, train the classifiers by feeding them the data produced and evaluate the testing data in order to be able to obtain meaningful and representative results.

So, we have defined the following specific goals:

- **Design a methodology to evaluate the effectiveness of intrusion detection algorithms**

  In order to conduct a systematic evaluation of container deployment, we aim to devise a rigorous conceptual methodology. This methodology must support and define the steps to follow during the evaluation procedure as well as be applicable to different containerisation platforms. In addition, it should be based upon an attack injection process given its proved application in other fields.

- **Study the stable definition of profiles from containers deployed in multi-tenant environments**

  As a crucial part of the evaluation procedure, it is necessary to define a stable and comprehensive profile of the container under monitoring. Therefore, we aim to study and understand through an exploratory procedure whether or not it is possible to

define a stable profile for containers through the use of the training data produced for this work.

- **Produce representative data to test container-based deployments**

  Due to a lack of available data for container-level deployments, it is required to produce data as a venue to evaluate the anomaly detection algorithms. Therefore, the definition of different and realistic usages of a container in order to produce data on the form of a trace, usable for training and testing purposes is of utmost importance.

- **Evaluate the effectiveness of host-based intrusion detection algorithms in containerised systems**

  Perform an experimental campaign to evaluate the effectiveness of state-of-the-art intrusion detection algorithms for container-based systems. This evaluation should report meaningful metrics in this context, such as recall and FPR, in order to extract representative results to support a decision to whether or not these intrusion detection is applicable to containers.

3.2 Relevant Threats to be Addressed

Intrusion detection has seen some developments, in terms of deployment and monitoring mode, such as previously referred distributed intrusion detection system - DIDS [15]. Despite great advances in intrusion detection for Virtual Machines (VMs)-based environments, container-level approaches have been neglected and work improvements in this context are still sparse.

As a result, this work emerges due to imminent threats to container-based multi-tenant cloud deployments [57]. The fact that various containers, with different owners, share physical components, raises the possibility that some tenants, with malicious intentions, may try to compromise others' execution, or even the normal execution of the host machine.

The most probable scenario is that there are no malicious activities developed to obtain access to the container placed inside the infrastructure. Only after obtaining access the container becomes malicious, performing mischievous activities and trying to compromise other tenants. The fact that cloud computing services assume their consumers are trustworthy allows an attacker to allocate resources lawfully, as any other customer, and use them. In other words, in this work, we assume that an agent gains access to computing resources within a cloud service lawfully and, only after, uses them to carry out illegal and malicious endeavours.

Therefore, our concerns rely upon the threats presented by these legally allocated resources, which originally are considered trustworthy despite existing the possibility of having malicious intention towards other tenants or the cloud service infrastructure. Which in fact stands a great menace to the underlying layers of the cloud service stack and to the normal execution of other lawfully obtained containers without wicked motivations.

Figure 3.1 depicts the threat model that is to be addressed during this work and that is described below. As it is possible to observe, we are particularly concerned with mali-

cious containers that share the infrastructure (multi-tenancy) with non-malicious one(s), performing three different types of attacks.

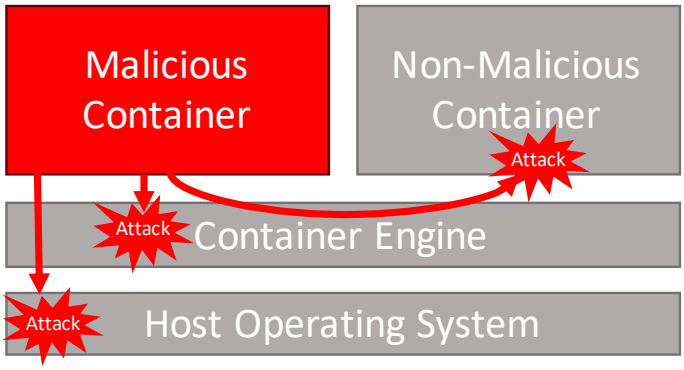


Figure 3.1: Container-based deployments' threat model.

Generally, every attack aims to compromise the infrastructure, or its components, and gain some type of reward, such as more resources or information to which the agent should not have access.

Regarding the attack to the host machine, a malicious container may try to exploit vulnerabilities present in the cloud's stack in order to attain access or gain control of privileged resources. Typically, these attacks are motivated by the ambition of collecting information or controlling the manner by which the resources are allocated.

Containers are commonly managed by an orchestration middle-ware, generally called a container engine. This also acts as an attack venue to achieve the Operating System (OS) layer, thereby this piece of software may also be compromised in cloud deployments by rogue containers, assuming the role of a pathway to achieve both of attacks, to the host machine and to other tenants. Hence, a malicious container would compromise the container engine in order to make its damage reach further. As a consequence, the attacks to the engine are a great concern owing to the damage that can come from it being compromised.

In sum, the major threat to cloud deployment services is the use of their infrastructure to conduct malicious activities against the infrastructure itself and the other clients who use it.

3.3 Research Approach

To accomplish the objectives outlined in Section 3.1, we defined a research approach consisting of three main phases. This approach is depicted in Figure 3.2 and described in the following paragraphs.

Initially, it is required to produce training datasets, collected for `10H` and `24H` in order to train classifiers and observe their evolution in the definition of a stable container profile. For this, we aim to periodically store the state of the normal behaviour database for later analysis, as explained in section 4.3.

Next, it is required to produce the datasets that are to be used in the evaluation procedure

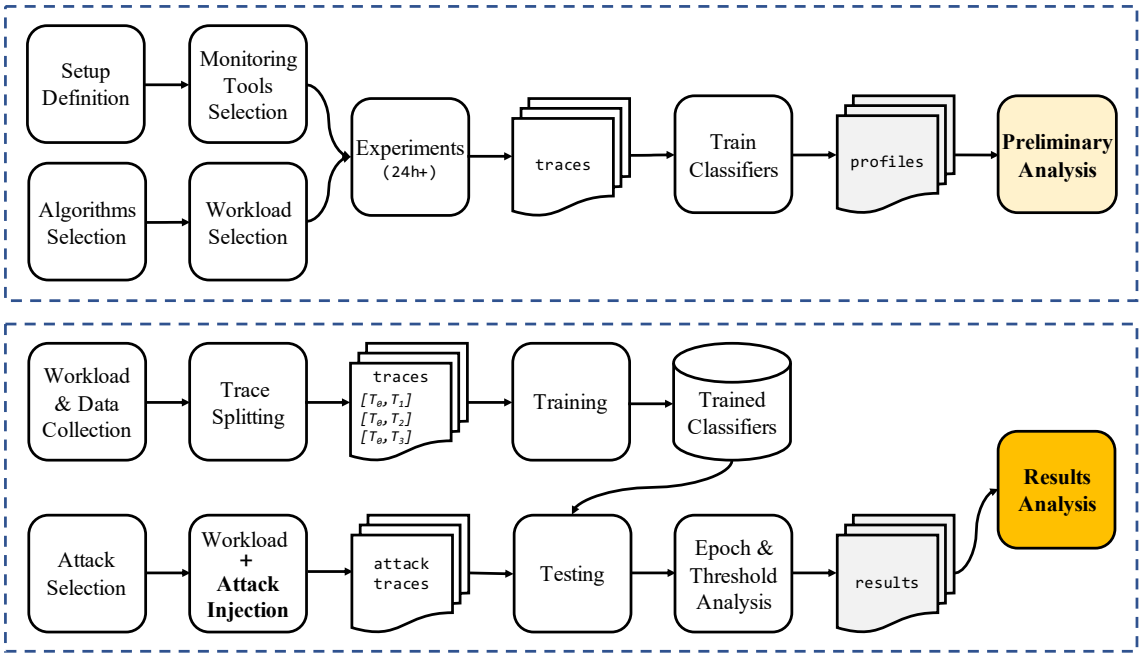


Figure 3.2: Overview of the followed research approach.

of the algorithms for intrusion detection. For this, we start through the collection of pure benign traces, from the monitoring of our container target. These traces were produced during a `24H` continuous monitoring and it was collected a set of attributes, such as date and time, system call issued, the number identifier (id) of the thread responsible for the invocation and the arguments passed to the function. In addition, the traces were generated for three platforms, OS, LXC and Docker containers, and for the two different types of workloads. More details regarding these datasets are provided in Section 4.4.

Still with respect to dataset generation, we were also required to generate the data utilised for testing the algorithms. Although this procedure is similar to the one conducted for training data generation, it differs in its time span and attack injection period. In order to obtain testing datasets, we conducted a procedure during which the target (container) was monitored for `30 minutes` while the workload was being applied and for each occasion, an attack took place at sensibly `15 minutes` after the start of data acquisition. This procedure was repeated for each one of the five attacks utilised, for each workload type and for each of the platforms.

Following the end of the data generation phase, we first conducted an exploratory study of the applicability of a set of algorithms to this context. During this study, we collected traces for the training procedure during `10H` and `24H`, using as workload an implementation of the TPCC On-Line Transaction Processing Benchmark [58] against a MariaDB server deployed at either a Docker or a LXC container. We utilised our implementations of Sequence Time-Delaying Embedding (STIDE) and Bags of System Calls (BoSC) to obtain profiles and assess whether or not they reached steady-state, that is a stable profile of the containers under monitoring. There was also an analysis of different window sizes, ranging from `3` to `6` where the number of entries in the normal behaviour database was monitored during the learning procedure.

Then, we conducted a systematised approach to the evaluation of the different host-based

anomaly detection state-of-the-art algorithms, namely BoSC, STIDE and Hidden Markov Models (HMM). During this procedure, multiple models for each algorithm were trained, specifically, for each training dataset were produced 12 classifiers for BoSC and STIDE and 3 classifiers for HMM. However, in total, there were 72 models produced for BoSC and STIDE, due to different window sizes, training times, training workloads and the platforms monitored whereas there were 6 models produced for HMM (only for Docker due to time restrictions), although they actually behaved as 24 classifiers after applying the 4 different thresholds to transform the probability value into a binary decision, either anomalous or normal. Therefore, overall there were 168 different classifiers trained for the evaluation phase.

The third phase of our research approach consisted of taking the trained classifiers and testing the datasets produced in the first phase in order to evaluate its performance. In this campaign, we tested the all BoSC, STIDE and HMM classifiers against the corresponding testing datasets and obtained the results. Lastly, the results were analysed with a epochs approach and the final results produced, which were then compared with the ground truth in order to compute the selected metrics.

This page is intentionally left blank.

Chapter 4

Preliminary Analysis

This chapter provides a preliminary analysis of the three main components of an intrusion detection procedure, namely the system under monitoring, the data collection tool and the analyser.

For each component, we conducted a preliminary analysis as a way to make informed decisions. Initially, the selection of the most appropriate monitoring tool from the first set of tools elicited based upon a monitoring evaluation focusing on the amount of data collected by each tool and a performance impact on the normal operation of the system under monitoring. Then, we conducted a preliminary analysis regarding the learning procedure of Sequence Time-Delaying Embedding (STIDE) and Bags of System Calls (BoSC) from which we could define with more accuracy and based on experimental results.

As we can observe, Figure 4.1 depicts the interaction between the multiple components. Firstly, there is a container, containing the services running within it, which is under monitoring through the utilisation of a tool capable of collecting the system calls issued by it, among other relevant data. Moreover, the sequence of system calls and data collected is stored as a trace for future use. The analyser, which consists of an instantiation of a algorithm for anomaly based intrusion detection, processes the data accordingly, and produces a normal behaviour database by keeping the relevant data to achieve a definition of containers profile. Finally, the analyser, when in detection mode, either produces or does not produce anomaly alerts.

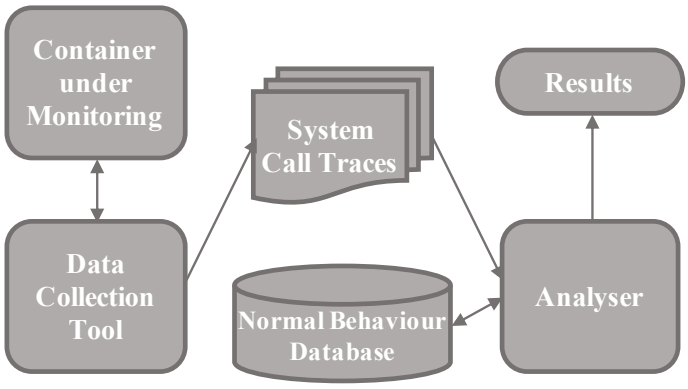


Figure 4.1: Architecture for the preliminary analysis of the intrusion detection applicability.

4.1 System Under Monitoring

With respect to the system under monitoring we established, based on previous work in container intrusion detection [16], the utilisation of MariaDB versions 8.0 and 5.5.28 depending on the procedure being executed to be monitored and profiled.

On the one hand, version 8.0 was deployed into a Docker and into a LXC container, in order to study the applicability of intrusion detection in this context, more precisely to study the capacity of the algorithms BoSC and STIDE to define stable profiles of each container (section 4.3).

On the other hand, version 5.5.28, which was selected due to its known vulnerabilities as they were required to perform the testing procedure, described in Section 4.4, was deployed in three different platforms. In addition to Docker and LXC deployments, this version of MariaDB was also deployed into a Kernel-based Virtual Machine (KVM), as a manner to emulate a OS-level deployment.

Furthermore, all the deployments were configured according to default MariaDB configuration. A database was created for each deployment and an implementation [1] of the TPC-C benchmark was utilised to set up the schema and load the data required for all experiments with 100 warehouses as main configuration.

4.2 Data Collection Tool

To fulfil this component, we selected two tools based on the applicability to our goals, which are containers monitoring and collection of the system calls issued by them.

The first tool we have elicited is ***strace*** [49], because it is present in most Linux systems and has the ability to attach to processes running on a machine, monitoring and collecting the system calls by them invoked along with the arguments passed to them.

Secondly, we selected the ***sysdig*** [46] utility, since it is a container monitoring and auditing specialised tool. It allows the collection of traces from containers containing data such as system calls and other Operating System (OS) events.

Both tools are capable of collecting system calls and other OS events, however, *strace* is process-oriented while *sysdig* is container-oriented, which means that, in our case, *sysdig* might be the best choice.

In order to select the most appropriate utility for the task in hands, we conducted some preliminary experiments to assess their applicability to containers monitoring.

Initially, we monitored the same container using both tools for a one-hour period. During this period, both tools stored the trace of system calls collected into separated datasets. Following this collection procedure, the system calls contained within each trace file were counted in a cumulative manner for each 10 minutes interval. The results obtained are presented in Figure 4.2.

Through the examination of Figure 4.2 is possible to observe that the *sysdig* tool is col-

[1] `https://github.com/Percona-Lab/tpcc-mysql`

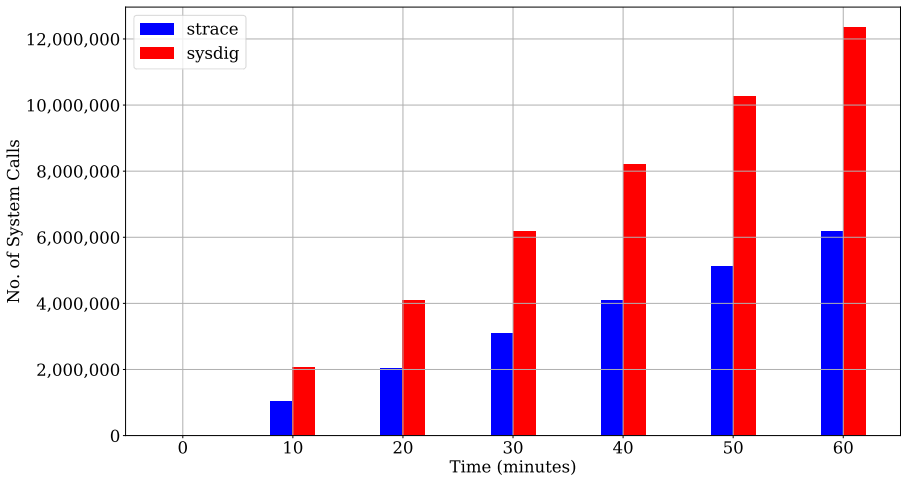


Figure 4.2: Comparison of system calls collected by each tool during 60 minutes.

lecting more system calls than *strace* in the same period of time for the same container. Although a small difference was expected, the difference observed at the end of the sixty minutes time period is huge and unexpected. Despite the differences observed in the amount of system calls, both tools monitor the exact same processes, therefore a closer and more thorough analysis is required to understand the reason behind the disparity on system calls amount.

In addition, we conducted an experiment to assess the impact of each data collection tool in terms of overhead and response time delay. For this experiment, we collected the results produced by the TPC-C benchmark when utilised against a LXC container, without a tool collecting the system calls, using *sysdig* or *strace*. Each configuration was repeated 3 times, having each one a period of two hours. This implementation of the TPC-C benchmark produces as output three measures of the response time of *New Order* transactions, the response time 95% percentile, the 99% percentile and the maximum response time registered during a given interval. The results were averaged for all the 3 repetitions and are presented in Table 4.1.

Table 4.1: Response time results for MariaDB when not monitored, monitored using *sysdig* and monitored using *strace*.

Tool	**95% Response Time**	**99% Response Time**	**Max Response Time**
Without Tool	1594.792	2171.075	2542.056
Sysdig	1727.007	2403.750	2858.827
Strace	1971.791	2749.419	3378.625

The response time values are in milliseconds (ms).

The analysis of the results obtained from TPC-C benchmark allows to comprehend the impact of both monitoring tools on the response time of the Database Management System (DBMS). Regarding the 95% percentile case, the response time showed a degradation of approximately 130 ms when the container was monitored using *sysdig* whereas the use of *strace* caused a degradation of approximately 400 ms. This gap grows even larger when we analyse the 99% percentile case, where *sysdig* demonstrates a degradation of nearly 230

ms while *strace* causes a 580 ms degradation in the response time. With respect to the average of the worst cases, the gap becomes even clearer for *strace*, where the response time differs from the without tool monitoring the container case by nearly 830 ms, while *sysdig* has a minor impact by raising the Max response time value by approximately 310 ms.

The main reason behind these results is the operation mode of *strace*. This tool, as explained in section 2.3.3, intercepts a system call, interrupting its execution collecting it, decoding it and resuming its execution. This mode of operation incurs in the addition of two new context switches for each system call [48], which results in the increase of the response time as verified on Table 4.1. Although the response time for *sysdig* also suffers an increase, it is less significant due to its less intrusive nature, which points out that *sysdig* is a better option to collect the system call traces from containers. Therefore, it is the tool which we are going to utilise during our practical experiments.

4.3 Analyser

In this section we focus on the component which analyses the data in order to either produce a profile of a container or the results of a trace analysis. The analyser receives input from the data collection tool in order to construct a container's profile.

In this preliminary analysis, we were focused on evaluating the evolution of the definition of a container's profile when using either BoSC or STIDE. For this, we collected benign traces from Docker and LXC containers running the MariaDB server application which received the workload produced by an implementation of the TPC-C On-Line Transaction Processing Benchmark [58], configured with 100 warehouses and using 50 clients during workload execution. We conducted two runs of collections for 10 and 24 hours of collection time period.

Table 4.2 provides an overview of the characteristics of the benign traces collected, containing the total number of system calls present in the trace as well as the number of unique ones. Docker traces seem to have on average a set of 35 unique system calls, although three out of four traces contain 37 unique system calls whereas LXC traces contain, on average, 125 unique system calls. These numbers differ greatly as a result of the type of containers we are monitoring. While docker containers are application containers, LXC containers are OS containers (see Section 2), therefore, docker containers have fewer processes running when compared to LXC, thus producing a less diverse set of unique system calls.

Table 4.2: Analysis of collected traces.

Training Time	**Platform**	**Run 1**		**Run 2**	
		Unique	**Total**	**Unique**	**Total**
10H	**Docker**	37	1,303,036,648	29	684,693,069
	LXC	116	3,279,400,309	127	2,962,761,415
24H	**Docker**	37	1,907,494,529	37	3,533,706,363
	LXC	129	9,617,350,813	129	10,305,731,175

Following data collection, we trained classifiers of the STIDE and BoSC methods with window size ranging from 3 to 6 system calls in each. We computed the slope value of the growth curve, which, in this context, represents the rate of new combinations (windows) of system calls added to the classifiers' normal behaviour database during a period of time.

For this, we used the formula $0 \leqslant \frac{S_{t_{s2}} - S_{t_{s1}}}{t_{s2} - t_{s1}} \leqslant \sigma$ [19] to decide whether an interval of the learning process is at learning steady-state. In addition, we consider that the steady-state of the learning procedure is achieved when the inequality above is satisfied 5 times in a row for $\sigma = 0.15$, based on previous experiments conducted on [19]. The results from these experiments are presented in Table 4.3 for Docker and on Table 4.4 for LXC.

Table 4.3: Docker results for reaching learning steady-state.

Training Time	**Algorithm**	**Window Size**	**Run 1**		**Run 2**	
			T_{max}(s)	**DB Size**	**T_{max}(s)**	**DB Size**
10H	**STIDE**	3	3,000	2,582	3,000	2,345
		4	20,700	21,952	24,100	18,625
		5	-	-	-	-
		6	-	-	-	-
	BoSC	3	1,400	1,486	1,900	1,406
		4	6,800	6,091	6,400	5,280
		5	20,700	19,066	20,800	16,273
		6	-	-	-	-
24H	**STIDE**	3	3,000	2,674	2,700	2,428
		4	30,900	22,317	29,800	22,091
		5	-	-	-	-
		6	-	-	-	-
	BoSC	3	1,700	1,649	1,500	1,460
		4	5,800	5,898	6,800	5,801
		5	26,400	19,214	23,300	18,544
		6	34,200	43,942	54,900	49,151

$T_{max}(s)$ represents when learning steady-state is reached, DB Size is the number of entries of the normal database at that moment.

The results are similar in terms of learning steady-state achievement for different platforms. The learning procedures converge to steady-state in approximately `3,000` seconds for STIDE when utilising a window of size 3 for Docker whereas for window of size 4 the required time raises to `26,500`. The reason behind the increase in the amount of time to achieve steady-state is due to the enlargement of the window size. On the case of BoSC, the convergence verified was on average `1,600` seconds for window of size 3, while for window of size 4 conducted to a convergence after, on average, `6,450` seconds of training. Unlike STIDE, BoSC also achieved a steady learning state for window 5 within the `10H` and, obviously, also for the `24H` of training time experiment. For this window size, the classifiers converged its training after, on average, `22,800` seconds of training. Furthermore, BoSC also achieves the steady state for classifiers with window 6 but only on the experiment where the traces of `24H` are utilised due to longer data available for training. In this case, we can observe that on average it takes around `44,550` seconds to achieve a stable profile for Docker with window 6 of BoSC.

With regard to LXC, the definition of profiles results, in general terms, in similar values to the ones obtained for Docker container in relation to the different windows that converged to a steady learning state.

In detail, there are two more entries in Table 4.4, which means that there are two more classifiers achieving learning steady-state for this case, namely the BoSC classifier with window 6 achieved a stable profile for the `10H` collection for run 2 while STIDE also achieved a stable profile with window 5 with the `24H` training trace for run 1.

Table 4.4: LXC results for reaching learning steady-state.

Training Time	Algorithm	Window Size	Run 1		Run 2	
			T_{max}(s)	DB Size	T_{max}(s)	DB Size
10H	STIDE	3	3,900	3,177	6,800	3,283
		4	10,800	15,692	9,600	15,106
		5	-	-	-	-
		6	-	-	-	-
	BoSC	3	3,900	2,376	6,800	2,487
		4	5,500	5,554	6,900	5,526
		5	13,800	13,979	19,300	19,783
		6	-	-	28,200	39,513
24H	STIDE	3	6,800	4,271	4,600	3,490
		4	12,100	19,563	24,300	26,312
		5	77,700	152,366	-	-
		6	-	-	-	-
	BoSC	3	4,800	2,857	4,600	2,644
		4	6,800	6,970	17,400	10,728
		5	12,100	16,867	24,300	24,037
		6	35,600	42,830	25,200	42,019

$T_{max}(s)$ represents when learning steady-state is reached, DB Size is the number of entries of the normal database at that moment.

STIDE took, on average, `5,375` seconds to reach a stable profile with window 3 while for window 4 the length of time required was around `14,200` seconds. In addition, it also achieved the stable profile for window 5 in one occurrence, for run 1 of 24H collection, where it took `77,700` seconds.

Regarding BoSC, the stable profile was achieved with every window size for almost all training traces, failing only for run 1 of the `10H` training traces. In this analysis, BoSC achieved a stable profile for window 3 after `5,025` seconds, while window 4 took on average `9,150` seconds. For window 5 it was registered an average time of `17,375` seconds and `29,700` for window 6.

Although it is no surprise that BoSC was, in general, faster than STIDE to achieve a stable definition of the profile, it was surprising the faster convergence for both algorithms on some window sizes for LXC when compared to Docker. This may indicate that LXC traces are less diverse than Docker traces despite of LXC generating more amounts of data and having a larger set of unique system calls within them 4.2. Another evidence of this is the achievement of a stable profile for LXC using STIDE with a window of size 5.

Figure 4.3 presents a visual display for the procedure executed for docker, while in Figure 4.4 we can analyse the procedure for the LXC container. In this figure, the blue dots represent the moments were the slope is below σ and the green dots represent the four preceding steady-state intervals to the interval where the classifiers reach the learning steady-state, represented by the red dot.

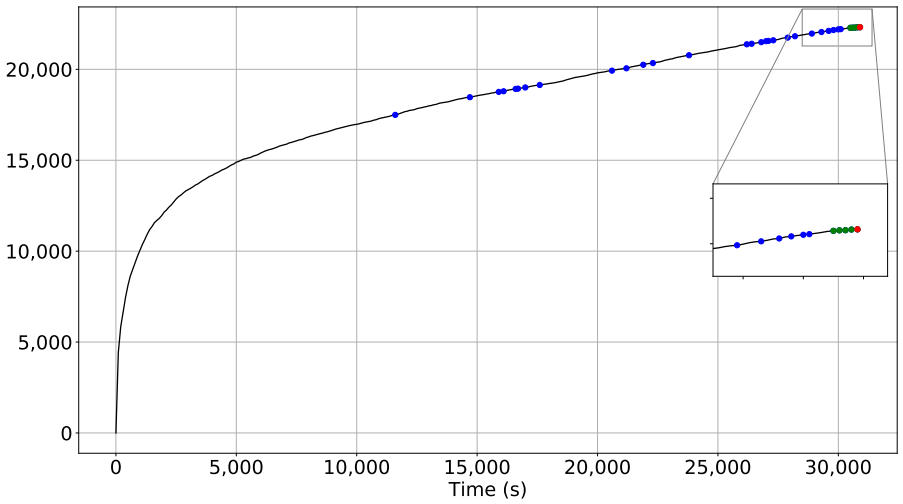


Figure 4.3: Training procedure for STIDE with window 4 of run 1 of 24h collection for Docker container.

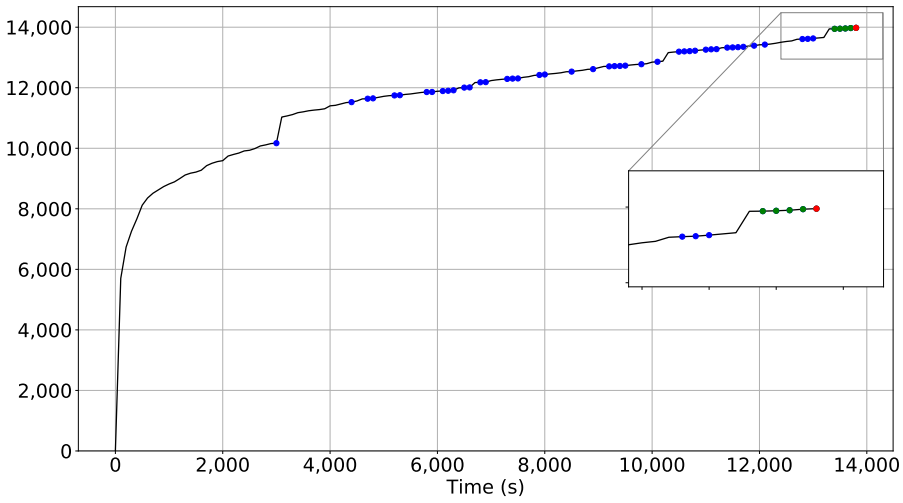


Figure 4.4: Training procedure for BoSC with window 5 of run 1 of 10h collection for LXC container.

In sum, the analysis of the results produced permits to conclude that, in the case of STIDE, we were able to achieve a learning steady-state using window size of 3 and 4, whereas, for BoSC it was possible to achieve learning steady-state with all window sizes despite requiring more training time for the larger windows. This means, that the faster configurations to achieve a learning steady-state, in this case, are the ones using windows of size 3 and 4. Moreover, the fact that STIDE did not achieve steady-state with windows of size 5 and 6, in most cases, is due to the fact that is a sequence-based method, theoretically having a larger set of possibilities for window combinations.

4.4 Datasets Analysis

In this section, we provide an analysis of the datasets produced during the experimental campaign (Chapter 5). To evaluate the effectiveness of intrusion detection algorithms on container-based systems, we produced 6 training datasets (2 for each platform: OS, LXC and Docker) of pure benign data and stored them in order to train the classifier. In addition, during the attack injection procedure, we collected traces of mixed data that is both benign and malicious data, in order test the trained classifiers. For each platform we collected 30 datasets, 3 for each exploit with one of the two workloads running resulting in a overall total of 90 testing traces.

Following, we provide an analysis of the datasets produced for each platform. The data presented is the result of the average of the three datasets for each vulnerability exploited.

Table 4.5: OS Datasets characteristics.

Workload	Vulnerability Exploited	Unique System Calls	Total System Calls	Time Length
Training				
WorkloadS		33	851,683,119	24:01:20
WorkloadN		33	694,952,987	24:00:29
Testing				
WorkloadS	CVE-2016-6662	38	18,505,855	30:22 min
	CVE-2012-5611	64	15,123,094	28:23 min
	CVE-2013-1861	67	15,192,527	27:23 min
	CVE-2012-5627	31	18,688,826	30:24 min
	CVE-2016-6663	35	19,242,165	30:20 min
WorkloadN	CVE-2016-6662	38	18,215,057	30:21 min
	CVE-2012-5611	65	14,221,510	28:02 min
	CVE-2013-1861	68	14,942,292	27:29 min
	CVE-2012-5627	28	18,022,340	30:31 min
	CVE-2016-6663	36	13,436,187	30:15 min

Firstly, the analysis of Table 4.5 permits to notice that the training traces has its basis upon a set of unique system calls with size 33. In addition, the dataset of *WorkloadS* contains a larger number of entries, approximately more 200 million than the dataset of *WorkloadN* although they only differed on nearly one minute.

With respect to the testing datasets, we can observe a larger set of unique system calls for the datasets corresponding to the exploitation of vulnerabilities, which cause the restart of the MariaDB server, namely vulnerabilities identified by CVE-2012-5611 and CVE-2013-1861. The exploitation of the other vulnerabilities resulted in a smaller number of unique system calls (a set of 38 for CVE-2016-6662 for both workloads), while vulnerability CVE-2012-5627 produced a group of 31 different system calls with *WorkloadS* and 28 with *WorkloadN*. This difference in the number of system calls is surprising, since it was expected a more diverse set of functions invoked with the non-steady workload due to the establishing and termination of connections related to its unstable nature. Nonetheless, the opposite

Table 4.6: LXC Datasets characteristics.

Workload	Vulnerability Exploited	Unique System Calls	Total System Calls	Time Length
Training				
WorkloadS		108	4,019,284,134	24:00:00
WorkloadN		112	3,110,046,388	24:02:20
Testing				
WorkloadS	CVE-2016-6662	86	163,651,213	30:21 min
	CVE-2012-5611	95	128,416,414	27:22 min
	CVE-2013-1861	93	120,542,507	27:18 min
	CVE-2012-5627	88	131,177,302	30:17 min
	CVE-2016-6663	97	108,606,545	30:17 min
WorkloadN	CVE-2016-6662	90	102,557,512	30:15 min
	CVE-2012-5611	97	91,653,675	27:38 min
	CVE-2013-1861	94	95,537,381	27:33 min
	CVE-2012-5627	88	115,406,453	30:49 min
	CVE-2016-6663	101	94,566,531	30:26 min

was verified. Furthermore, the exploitation of CVE-2016-6663 resulted in a similar amount of different functions invocation (35 with *WorkloadS* ; 36 with *WorkloadN*). Finally, all the datasets are constituted by a total number of system calls in the range of `13 million` to *20 million*.

With regard to the datasets produced by LXC containers, Table 4.6, the observation of the results for training datasets immediately allows to highlight the expected difference between traces from *WorkloadS* and *WorkloadN*. As explained in Section 5.1.1, *WorkloadN* is a more unstable workload in terms of active connections oscillation, thus it was expected to produce a larger variety of system calls.

Regarding testing traces in general, we can observe the expected behaviour of a larger set of system calls for datasets of *WorkloadN*, although not very significant. For CVE-2016-6662, a set of 86 functions was observed for the steady workload whereas *WorkloadN* produced, on average, 90 different system calls. The traces of the exploitation of CVE-2012-5611 and CVE-2013-1861 produced similar amounts of system calls while the exploitation of CVE-2016-6663 produced the larger set of unique functions invoked, 97 for *WorkloadS* and 101 for *WorkloadN*. Lastly, by executing the Proof of Concept (PoC) for CVE-2012-5627, both types of traces produced on average 88 different system calls.

From these results stood out the amount of total system calls produced for each type of collection. While for training datasets it was registered a number of 4 billion for steady workload and 3 billion for the non-steady workload, the testing datasets ranged, in length, from 91 million to 164 million. These large amounts might be justified by the fact that LXC is an OS containers and has multiple services running at the same time.

The analysis of Table 4.7 provides the comprehension of Docker datasets' characteristics. Unlike previous datasets, from OS and LXC, these traces are clearly distinct and more diverse between the different workloads. The training traces denote a disparity of 28 system

Table 4.7: Docker Datasets characteristics.

Workload	Vulnerability Exploited	Unique System Calls	Total System Calls	Time Length
Training				
WorkloadS		38	687,013,250	47:59:41
WorkloadN		66	786,170,413	48:00:18
Testing				
WorkloadS	CVE-2016-6662	36	8,107,429	30:34 min
	CVE-2012-5611	88	6,673,828	27:23 min
	CVE-2013-1861	88	6,483,013	26:19 min
	CVE-2012-5627	24	7,865,766	31:25 min
	CVE-2016-6663	80	7,739,002	30:29 min
WorkloadN	CVE-2016-6662	57	7,507,516	30:31 min
	CVE-2012-5611	88	6,956,954	28:28 min
	CVE-2013-1861	89	6,227,981	28:32 min
	CVE-2012-5627	35	7,251,557	30:27 min
	CVE-2016-6663	80	6,085,046	30:33 min

calls between them. While the trace from *WorkloadS* contains 38 distinct functions, the trace from *WorkloadN* comprehends 66 unique system calls. The length of the datasets range from 687 to 786 million for steady and non-steady workloads, respectively.

Regarding the testing traces, the same disparity remains in datasets which resulted from the exploitation of vulnerabilities CVE-2016-6662 and CVE-2012-5627 when comparing *WorkloadS* to *WorkloadN*. However, the remaining vulnerabilities, when exploited, resulted in similar amounts of distinct functions. For CVE-2012-5611 88 system calls detected in both traces, while 80 in both traces for CVE-2016-6663 and 88 and 89 for CVE-2013-1861 with steady and non-steady workloads, respectively. The amounts of total data ranged from 6 to 8 million.

Figure 4.5 provides a visual comparison of the amount of distinct system calls in each platforms' datasets per vulnerability and training. The depicted bar chart intends to provide a clearer and easier comparison between the different platforms and vulnerabilities as a manner to explicitly understand the diversity of data among the different deployment methods.

Although unexpectedly, and as previously observed, the differences between traces collected from *WorkloadS* and *WorkloadN* are not very significant. Despite Docker producing clear-cut differences, the same could not be observed for either LXC or OS. The observation of the chart denotes a clear domination of LXC in terms of distinct system calls amounts, being the more productive in terms of distinct system calls, for every trace.

Typically Docker produces more diverse datasets than OS, although, in specific situations such as exploitation of CVE-2016-6662 and CVE-2012-5627 with *WorkloadS*, the opposite is verified.

In sum, despite OS datasets generate greater amounts of system calls, they are, in average,

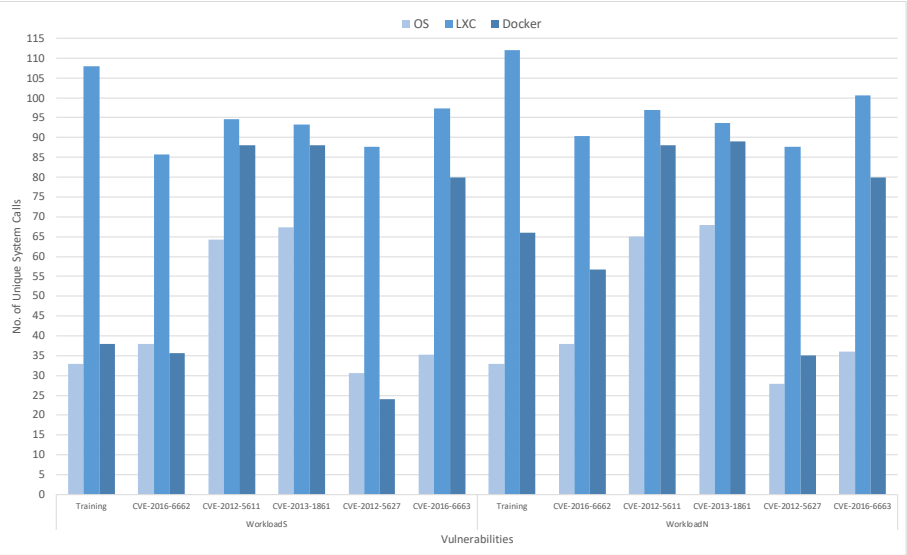


Figure 4.5: Comparison of unique system calls registered during experiment datasets creation.

less diversified than Docker traces, which despite fewer total system calls contain more unique ones. For LXC occurs a higher diverse number of functions as well as a heavily higher number of total system calls.

This page is intentionally left blank.

Chapter 5

Evaluating Intrusion Detection Algorithms in Containerised Systems

In this chapter, we focus on the experimental evaluation of state-of-the-art and widely used anomaly-based intrusion detection algorithms, very utilised in other contexts, for container-based systems. We start by providing an overview of the proposed experimental methodology utilised to evaluate the effectiveness and applicability of intrusion detection algorithms to container-based systems based upon representative and meaningful metrics. In addition, we explain the attack injection procedure and provide details regarding the vulnerabilities selected to use during this process as well as the experimental procedure itself. We also provide explanation regarding the analysis of sequences/windows from the classifiers results, explaining the epoch analysis and *thresholding* mechanism. Lastly, we present the results obtained and discuss from different perspectives.

5.1 Experimental Methodology

Our main goal is to understand whether or not intrusion detection techniques are applicable in container based systems. We are particularly interested in multi-tenant scenarios, where one of the tenants exploit vulnerabilities of his neighbour containers or their software, to attack the infrastructure and the remaining containers.

For this, we designed an experimental methodology based on attack injection, trying to understand if the intrusion detection algorithms provide interesting results in this context, and also trying to understand the impact of the different configuration of the algorithms in the results. Figure 5.1 presents an overview of this methodology.

Initially, we used one of the workloads against the system setup in order to generate the training traces. These were further used to provide the algorithms with data in order to build profiles for the corresponding platform. The traces were used incrementally, for Bags of System Calls (BoSC) and Sequence Time-Delaying Embedding (STIDE) we store the classifiers at `6H`, `12H` and `24H` whereas for Hidden Markov Models (HMM) due to memory constrains we trained these classifiers for `30min`, `1H` and `2H`. These classifiers were stored for later use.

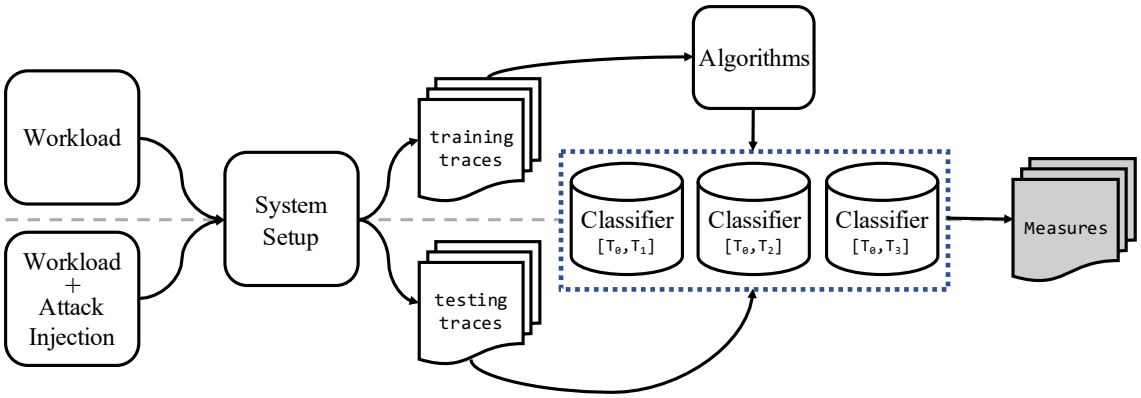


Figure 5.1: Overview of the proposed experimental methodology.

Following we conducted the attack injection procedure, by using one of the workloads at a time and injecting the attack at sensibly the middle of the time interval. This results in the production of mixed type traces, with benign and malicious data to be used for testing the classifiers.

Thus, following the production of testing traces and the classifiers training, the testing phase takes place, through feeding the testing data to the classifiers and generating the measurements.

5.1.1 Workload Characterisation

In order to perform a representative evaluation of the techniques under test, we used two different types of workloads. For this, we used an implementation of the TPC-C transaction processing benchmark [58] with two variations as a manner to emulate distinct usage profiles. The two workloads are characterised as follows:

- **WorkloadS** is a **steady workload** with 50 clients connected to the server, continuously performing the TPC-C transactions.
- **WorkloadN** is a **non-steady workload** with an interval of clients ranging from 10 to 90. The workload starts with 10 active connections, and increases the number of clients by 8 every 3 minutes until it reaches the maximum number of connections. The life span of each connection is 30 minutes, thus after reaching its peak, it starts decreasing until 10, again. The process repeats for each hour. In addition, the time parameters previously referred are configurable, as a way to have the same behaviour during the collection of testing datasets.

These workloads were selected to provide two very diverse operational profiles, to help us understand the impact that the type of workload has in the results.

5.1.2 Attack Injection

To assess the detection capabilities of the algorithms under test, we must perform an attack injection procedure so that we are able to collect the traces produced from these malicious

endeavours. For this, we searched the vulnerabilities of the MySQL and MariaDB Database Management System (DBMS) aiming to find their Proof of Concept (PoC), that is exploits.

We aimed at collecting diverse PoC in terms of consequences, type of attack and exploitation procedure. Therefore, after compiling a set of different PoCs, we selected the ones which provide us with, both remote and local exploits, through buffer overflows, password cracking, code execution and condition racing, as a venue to achieve privilege escalation or cause a denial of service.

As a result, the compiled list of Common Vulnerabilities and Exposure (CVE)s corresponding to each of the collected 5 working PoCs whose details are presented in Table 5.1. Further information about each vulnerability is provided in the following sub-sections.

Table 5.1: List of vulnerabilities used and respective CVE information.

CVE ID	Access Type	Vulnerability Type(s)	Score	PoC Reference
CVE-2012-5611	Remote	Execute Code, Overflow	6.5	[59]
CVE-2012-5627	Remote	Execute Code	4.0	[60]
CVE-2013-1861	Remote	Denial Of Service, Overflow	5.0	[61]
CVE-2016-6662	Remote	Execute Code, Bypass	10.0	[62]
CVE-2016-6663	Local	Gain privileges	4.4	[63]

CVE-2012-5611

The vulnerability with the CVE-2012-5611 allows remote authenticated users to execute arbitrary code via a long argument to the GRANT FILE command which causes a stack-based buffer overflow in the `acl_get` function and as a consequence it causes the server to restart [64]. ThePoC, which we collected from [59], exploits this vulnerability by trying to grant file permissions to a database with a name length of 100,000 characters. This name length causes a buffer overflow and forces the server to restart.

CVE-2012-5627

The CVE-2012-5627 announces that MariaDB does not modify the salt during multiple executions of the `change_user` command within the same connection which makes possible for remote authenticated users to conduct brute force password guessing attacks [65]. The PoC, originated from [60], connects to a database and through the use of a password cracking tool, *John, the Ripper* [1], which due to a constant salt value it is effective.

CVE-2013-1861

CVE-2013-1861 states that the server allows remote attackers to cause a denial of service via a crafted geometry feature that specifies a large number of points, which is not properly

[1] `https://www.openwall.com/john/`

handled [66]. Therefore, the PoC, collected from [61], exploits this by providing a large value, and successfully crashes and forces the server to restart.

CVE-2016-6662

Regarding the vulnerability identified by CVE-2016-6662, it allows users to create arbitrary configurations and bypass certain protection mechanisms by setting `general_log_file` to a `my.cnf` configuration [67]. This vulnerability is classified with a Common Vulnerability Scoring System (CVSS) score of 10.0, that is the highest score of the scale and the highest in our set of vulnerabilities. The PoC (from [62]) we collected, modifies the log file to `my.cnf` and injects a change in it, adding a line to force the loading of a dynamic library after a restart in the DBMS, which proceeds to escalate privileges and provide the attacker access to a root shell. However, during our experiments, we did not restart the server and only modified configurations and uploaded the dynamic library, which is actually the attack led by the attacker.

CVE-2016-6663

The vulnerability identified by CVE-2016-6663 allows local users with certain permissions to gain privileges, through a racing condition attack, through leveraging use of `my_copystat` by `REPAIR TABLE` to repair a `MyISAM` table [68]. The PoC from [63], tries to alter the permissions of a given file in order to win the race conditions and, ultimately, assign the intended permissions to a copy of the shell file as a way to achieve a shell with the privileges of the `mysql` user.

5.1.3 Experimental Procedure

The experimental procedure is divided into two stages, analogous to the operating stages of an anomaly-based Intrusion Detection System (IDS). Thus, the procedure consists of a training stage and a testing stage, which resulted in the production of the datasets analysed on Section 4.4. During the training stage, the following procedures were conducted:

1. **Loading phase**: 16 million insert operation.
2. **Collection phase**: collect the system calls issued by the container during `24h`.
3. **Training phase**: train the algorithms using the data gathered

With focus on the testing stage, it is conducted according to the following steps:

1. **Transaction phase**: start one of the two workloads during `30min`
2. **Injection phase**: inject the attack at the determined moment
3. **Testing phase**: feed the testing data to the trained algorithm
4. **Measuring phase**: the results were compared with the ground truth and the measurements performed

In addition, we had to select some configurations to use during the testing phase, namely, the different training times, `6h`, `12h` and `24h`, the distinct sizes for the window used by the algorithms, those are `3`, `4`, `5` and `6`, the number of windows per epoch, `500`, `1000` and `5000` and the value for the detection threshold for which we selected `5`, `10`, `20`, `50` and `100`. All the values presented were used for BoSC and STIDE whereas the use of HMM required some slight variations. Specifically, the training time was quite short due to spacial restrictions, in this case, we defined training times of `30min`, `1h` and `2h`. Moreover, the window sizes do not apply to this algorithms, therefore are not considered, however, these classifiers require another configuration parameter, the threshold utilised to decide whether or not a sequence is anomaly. Thus, for this parameter, we selected the values of `50`, `100`, `125` and `150`, after an examination of the preliminary results produced. All other parameters remained utilised with the same values as for BoSC and STIDE. Trying to mitigate the impact of the non-determinism of this type of experiments, each slot was repeated 3 times.

5.1.4 Measures

The experiments conducted were characterised through the gathering of information about the classification produced by the trained algorithms. More concretely, we stored the initial results produced by the classifiers, that is, the direct result of the window classification performed by BoSC and STIDE, and the sequence analysis conducted by HMM. While the first two produced a final result for each window (normal or anomalous), HMM produced the probability of a sequence belonging to the model, that is, the probability of having been issued by the container to which the profile belongs. As a consequence, in the case of HMM, there is an additional step, which consists of applying the thresholds defined above, in order to produce a final result in the same manner as BoSC and STIDE. These final results produced during this step are stored for later processing.

Following the previous analysis, we conducted an evaluation of the results produced based upon an epoch analysis where the parameters epoch size and detection threshold are enforced, as depicted in Figure 5.2. So, the results are fragmented into epochs of various lengths, according to the value of the parameter and these analysed through the frequency of each possible output (normal or anomalous) within that epoch. After performing the number of occurrences of each class, the value is compared against the value of the parameter detection threshold. Then, when the frequency of anomalous windows/sequences is greater or equal to the value of the detection threshold, the epoch under analysis is considered anomalous.

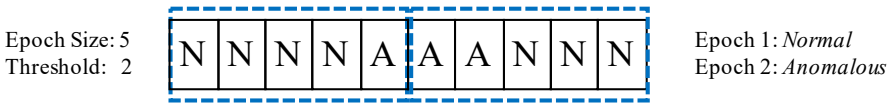


Figure 5.2: Example of epoch analysis of the results.

At the end of this procedure, we conduct a comparison of the results obtained for each classifier against the ground truth thus computing the absolute values for True Positives (TPs), False Positives (FPs), True Negatives (TNs) and False Negatives (FNs), which are later used to compute the performance metrics defined in Section 2.2.2.

We computed the value of recall, through the use of TP and FN in order to verify whether the classifiers detect the existing attacks, as well as the value for precision to assess how precise the results produced are by using TP and FP and FPR to understand the percentage of FP alarms raised among the true anomalies present in the traces.

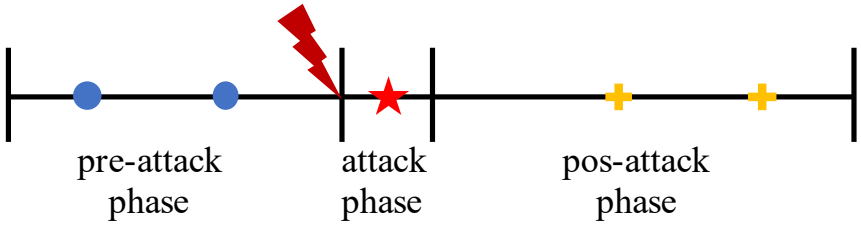


Figure 5.3: Attack slot classification.

Moreover, we divided the epoch analysis period into three sections, the pre-attack phase, the attack phase and the post-attack phase because these sections are important due to their relation to the computation of the metrics values. Reports of anomalous epochs registered within the attack phase were classified as TPs ($\star$), while the remaining were classified as FPs. If no epoch was reported as anomalous during the attack phase, we classify it as FN.

Finally, the analysis of the performance of classifiers is based upon the metrics computed through these four absolute values.

5.1.5 Experimental Campaign

This section presents the experimental campaign which follows the methodology described in the previous section. In this campaign, we conducted an analysis on methods used for intrusion detection in other contexts [69], namely BoSC and STIDE, regarding their performance on the container-level context. More precisely, we deployed a MariaDB server (version 5.5.28) onto a Kernel-based Virtual Machine (KVM), a Docker and a LXC container on a machine and performed a series of experiments.

$$\textit{3 training times} \times \textit{2 training workloads} \times \textit{2 algorithms} \times \textit{4 window sizes} \times \textit{5 exploits} \times \textit{2 testing workloads} \times \textit{3 epoch sizes} \times \textit{5 detection thresholds}$$

We also conducted a similar experiment for HMM, however, just for the Docker platform.

$$\textit{3 training times} \times \textit{2 training workloads} \times \textit{1 algorithm} \times \textit{4 decision thresholds} \times \textit{5 exploits} \times \textit{2 testing workloads} \times \textit{3 epoch sizes} \times \textit{5 detection thresholds}$$

The use of KVM aims to represent a traditional deployment of an application, in this case MariaDB, at the OS level, as a manner to provide basis for comparison against container deployments in order to validate the results obtained. In addition, the use of KVM allows to reset the state of the system without the need to perform a clean installation of an OS.

Figure 5.4 depicts an overview of the tests slots, each test slot consists in a sequence of three collections of the relevant data from the monitoring target. Each collection consists

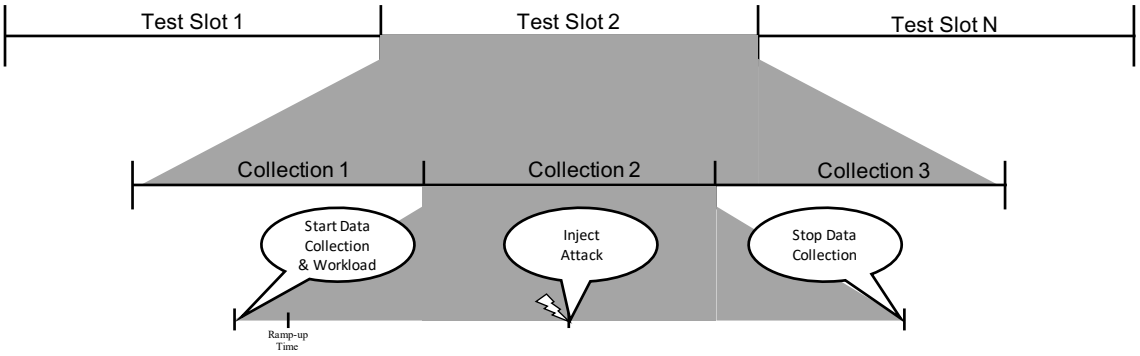


Figure 5.4: Experimental procedure and test slots.

in starting the collection of data and the execution of a workload, and an injection attack procedure noticeably at the middle of the time period, which in practice means an injection of the attack at `15min` after the start of the collection period, since each one lasts for `30min` that is when the last action is performed, to stop the collection of data.

The Figure 5.5 depicts the setup used during this experimental procedure whose constituents were a Test Driver and a Container Infrastructure machines.

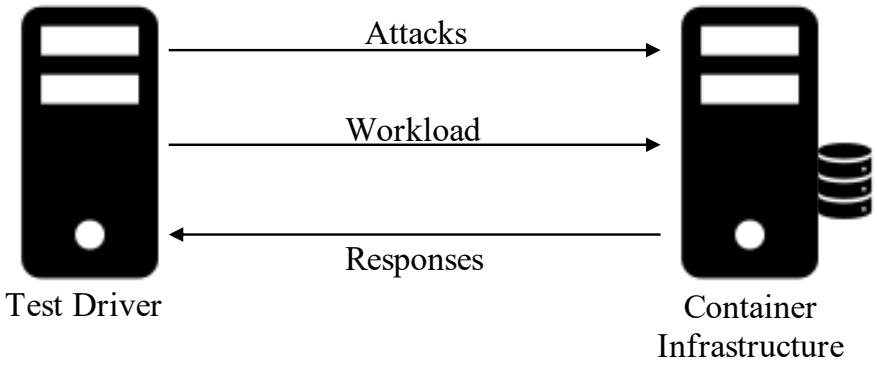


Figure 5.5: Setup utilised on the experimental campaign.

The Test Driver is provided with an Ubuntu 16.04.3 LTS 64 bit installation, while taking advantage of a 3.20GHz CPU, 16GB of RAM and a SSD and HDD. Additionally, this machine was responsible for executing the training workload and the testing workload as well as the remote attacks and collect the interval period values to establish the ground truth for further analyses. The Container Infrastructure machine had installed a 64 bit version of Ubuntu 18.04.2 LTS making use of a 2.8GHz CPU with 32GB of RAM and SSD and HDD, wherein the containers under test were located. The traces of system calls generated by the containers were collected and stored in this machine, for later processing.

To collect the traces of system calls, we utilised the container monitoring tool *sysdig* [46], as decided previously. Sysdig is a container-oriented tool that natively supports containers and allows users to format its output as intended. The containers used in this experimental campaign was configured using the Ubuntu 14.04 LTS Operating System (OS) and then the MariaDB version 5.5.28 was installed on it from the sources available at their GitHub repository [2]. The MariaDB server was installed with its default configurations.

[2] `https://github.com/MariaDB/server`

5.2 Results and Discussion

In this section we present and discuss the results obtained during the experimental campaign. We analyse the results produced for the three platforms, OS, LXC and Docker. The results are analysed from multiple perspectives, such as training time or training workload utilised, and throughout the course of the analysis of results some options are selected, based on observations, namely, truncating the results under analysis according in order to select the most interesting results.

We start through the analysis of the results produced by classifiers trained with different workloads and various training times, and then proceed to perform a more focused analysis, initially, concentrating upon window size, for BoSC and STIDE, and decision threshold for HMM and then continuing on to analyse the impact of different epoch sizes and detection thresholds based on its expected cost.

In addition, we also reflect upon the classifiers' generalisation capacity, that is, the capacity of detecting intrusions when testing datasets which use a different workload from the one with which were trained. Moreover, an analysis from the exploits perspective is also provided as well as an overview of the percentage of anomaly reports from classifiers according to the phase (see Figure 5.3 of the testing dataset.

The results presented come from the average of the three attack injection procedures within each test slot to reduce the impact of the non-determinism inherent to these experiments.

5.2.1 Overall Results for all Platforms

In this section, we provide an overview of the results. Table 5.2 provides a synopsis of the general results, which are fully presented in Appendix A.1.

For this overall analysis, we anchored some parameters as a way to truncate the number of results thus making them more presentable for this section. The observation of data presented permits to already perceive interesting results, as for instance, the high *precision* values for both BoSC and STIDE when testing for intrusions against OS deployment, despite lower results for *recall* when compared to the values obtained in the case of LXC.

For this platform, the results are also pretty satisfactory, where STIDE even reaches `0.999` for *precision*, for window of size 3, epoch of size 500 and detection threshold of 100, also achieving a *F-Measure* of `0.874` for the same configuration. Nonetheless, there are also unsatisfactory results, such as a *precision* of `0.285` for STIDE with window of size 5, epoch size of 5000 and a detection threshold of 100, nevertheless, it stills achieves a very high *recall* value of `0.966`.

With regard to Docker, there are as well some oscillations, with *recall* ranging from `0.588` to `1.000`, whilst *precision* varies between `0.471` and `0.914` across the different algorithms utilised.

Furthermore, in the following sections, we dive deeper into parameter-focused analyses aiming to identify the best configuration which achieves the best results in the cost-effectiveness paradigm.

Table 5.2: An overview of the results for all platforms, with 24H training time for BoSC and STIDE and 2H for HMM.

Platform	Algorithm	Window / Decision Threshold	Epoch Size	Detection Threshold	Recall	Precision	F-Measure	FPR
Docker	BoSC	3	500	20	0.982	0.878	0.927	0.003
				100	0.975	0.914	0.944	0.002
			5000	20	1.000	0.613	0.760	0.016
				100	0.970	0.876	0.920	0.003
		5	500	20	1.000	0.832	0.908	0.005
				100	0.975	0.912	0.943	0.002
			5000	20	1.000	0.534	0.696	0.022
				100	0.970	0.826	0.892	0.005
	STIDE	3	500	20	0.987	0.869	0.924	0.004
				100	0.975	0.914	0.944	0.002
			5000	20	1.000	0.592	0.744	0.018
				100	0.970	0.870	0.917	0.004
		5	500	20	1.000	0.809	0.894	0.006
				100	0.975	0.904	0.938	0.003
			5000	20	1.000	0.508	0.673	0.025
				100	0.970	0.747	0.844	0.008
	HMM	100	500	20	0.796	0.868	0.830	0.003
				100	0.625	0.892	0.735	0.002
			5000	20	0.840	0.486	0.616	0.032
				100	0.787	0.702	0.742	0.012
		150	500	20	0.654	0.844	0.737	0.003
				100	0.588	0.891	0.708	0.002
			5000	20	0.707	0.471	0.565	0.029
				100	0.653	0.676	0.664	0.011
LXC	BoSC	3	500	20	0.820	0.985	0.895	0.000
				100	0.603	0.999	0.752	0.000
			5000	20	0.914	0.664	0.769	0.004
				100	0.819	0.991	0.897	0.000
		5	500	20	0.977	0.672	0.796	0.004
				100	0.876	0.978	0.924	0.000
			5000	20	0.977	0.314	0.475	0.016
				100	0.963	0.487	0.646	0.008
	STIDE	3	500	20	0.833	0.963	0.893	0.000
				100	0.776	0.999	0.874	0.000
			5000	20	0.942	0.533	0.681	0.007
				100	0.827	0.973	0.894	0.000
		5	500	20	0.979	0.577	0.726	0.006
				100	0.943	0.840	0.889	0.002
			5000	20	0.979	0.285	0.442	0.021
				100	0.966	0.390	0.555	0.013
OS	BoSC	3	500	20	0.629	0.977	0.766	0.000
				100	0.593	0.987	0.741	0.000
			5000	20	0.664	0.876	0.755	0.001
				100	0.586	0.962	0.728	0.000
		5	500	20	0.666	0.935	0.778	0.000
				100	0.593	0.985	0.740	0.000
			5000	20	0.670	0.703	0.686	0.003
				100	0.594	0.901	0.716	0.001
	STIDE	3	500	20	0.635	0.972	0.769	0.000
				100	0.593	0.987	0.741	0.000
			5000	20	0.670	0.834	0.743	0.001
				100	0.586	0.953	0.726	0.000
		5	500	20	0.666	0.886	0.760	0.001
				100	0.593	0.981	0.739	0.000
			5000	20	0.670	0.631	0.650	0.004
				100	0.594	0.826	0.691	0.001

5.2.2 Train Time Impact Analysis

For this analysis, we selected the results produced by the classifiers in function of their training time. There are three different training times for each classifier, whilst BoSC and STIDE models were training for `6H`, `12H` and `24H`, HMM models were fed with `30min`, `1H` and `2H` of training data.

These results (Table 5.3) already allow to comprehend a general better performance, in terms of *recall* for algorithms operating on the containers domain. That is, while for OS deployment mode the value of *recall* ranges from `0.649` to `0.675`, for both LXC and Docker all values, with the exception of HMM, are above of `0.930`, which may indicate that either the attack patterns stood out more or the profile is not as stable for LXC and Docker as it is for the OS deployment once *FPR* is also higher for the container-based deployments.

In addition, it is very clear that, on general terms, HMM perform very poorly, achieving a *precision* values in the range of `0.106` to `0.117`, however, it still achieves a *recall* which ranges from `0.753` to `0.845`. This is a clear indication that HMM models are, in general terms, raising a high number of anomalies from which a great number is without reason as established by the high value of *FPR*.

Table 5.3: Training Time Analysis for all classifiers for all platforms.

Platform	Train Time (H)	Algorithm	Recall	Precision	F-Measure	FPR
Docker	6	BoSC	0.990	0.782	0.874	0.007
		STIDE	0.990	0.722	0.835	0.009
	12	BoSC	0.990	0.786	0.876	0.007
		STIDE	0.990	0.742	0.848	0.009
	24	BoSC	0.989	0.789	0.878	0.007
		STIDE	0.990	0.754	0.856	0.008
	30min	HMM	0.753	0.106	0.186	0.174
	1	HMM	0.769	0.109	0.191	0.173
	2	HMM	0.845	0.117	0.206	0.175
LXC	6	BoSC	0.959	0.563	0.709	0.006
		STIDE	0.972	0.474	0.637	0.009
	12	BoSC	0.950	0.609	0.742	0.005
		STIDE	0.965	0.521	0.677	0.008
	24	BoSC	0.935	0.661	0.775	0.004
		STIDE	0.956	0.585	0.726	0.006
OS	6	BoSC	0.666	0.845	0.745	0.001
		STIDE	0.675	0.686	0.680	0.003
	12	BoSC	0.657	0.869	0.748	0.001
		STIDE	0.667	0.767	0.713	0.002
	24	BoSC	0.649	0.894	0.752	0.001
		STIDE	0.658	0.812	0.726	0.001

From the training time point of view, there is, nonetheless, a general, clear and undeniable improvement in the *precision* values obtaining as well as the *FPR* registered with the

increase of the training time. This affirmation is corroborated through the evolution of *F-Measure* values, as its increase is motivated by the increase of *precision* and stable values regarding *recall*. Therefore, it is possible to affirm the increase of training time produces improved results.

Summary

- *Precision* and, therefore, *F-Measure* results improve overtime
- The most appropriate training time for both BoSC and STIDE is the `24H` period.
- The most appropriate training time for HMM is the `2H` period.

5.2.3 Train Workload Type Impact Analysis

In this section, we perform an evaluation from the training workload perspective, aiming to comprehend whether or not there is a workload which provides more information thus allowing to define a more stable profile of the container/application under monitoring and, therefore, achieve higher detection rates. In this analysis, the results presented on Table 5.4 are filtered by training time after the analysis from the time perspective (see Section 5.2.2), where concluded the `24H` and `2H`, for HMM, periods provide the most satisfactory results.

In connection with *recall* values, we can observe high and constant results, even though OS deployment mode is in the region of `0.638-0.664`, which are quite lower results when compared to either LXC or Docker (`0.928-0.958` and `0.807-0.990`, respectively). While *precision* values remain in the range of 60-90%, in general, and *FPR* does not achieve 1% for BoSC and STIDE, HMM models denote low *precision* values and high *FPR*.

Table 5.4: Training Workload analysis for classifiers trained during 24H for BoSC and STIDE, and 2H for HMM, for all platforms.

Platform	Train Workload	Algorithm	Recall	Precision	F-Measure	FPR
Docker	WorkloadS	BoSC	0.990	0.763	0.862	0.008
		STIDE	0.990	0.730	0.840	0.009
		HMM	0.807	0.080	0.145	0.256
	WorkloadN	BoSC	0.989	0.818	0.895	0.005
		STIDE	0.990	0.779	0.872	0.007
		HMM	0.883	0.205	0.333	0.094
LXC	WorkloadS	BoSC	0.941	0.693	0.798	0.004
		STIDE	0.953	0.604	0.740	0.005
	WorkloadN	BoSC	0.928	0.626	0.748	0.004
		STIDE	0.958	0.567	0.713	0.006
OS	WorkloadS	BoSC	0.660	0.883	0.756	0.001
		STIDE	0.664	0.809	0.729	0.001
	WorkloadN	BoSC	0.638	0.905	0.748	0.001
		STIDE	0.651	0.814	0.723	0.001

Nevertheless, from the perspective of training workloads' impact, it clear that for OS and Docker, the transition from *WorkloadS* classifiers to *WorkloadN* classifiers allows them to augment their *precision* (e.g., from `0.883` to `0.905`, BoSC for OS) and reduce the *FPR* (e.g., from `0.008` to `0.005`, BoSC for Docker) without compromising *recall*. However, unexpectedly, in the case of LXC, the classifier whose results were better, were the ones trained with *WorkloadS*. Producing higher *recall* (e.g., from `0.928` to `0.941`, BoSC for LXC) and *precision* (e.g., from `0.567` to `0.604`, STIDE for LXC) values while also slightly reducing *FPR* (e.g., from `0.006` to `0.005`, STIDE for LXC).

These observations provide reason to state that the *WorkloadN* provides more satisfactory results than *WorkloadS*, for the case of OS and Docker whereas *WorkloadS* produces better results for LXC.

Summary

- The training workload utilised has influence over the results obtained.
- For LXC, results show that *WorkloadS* permits to achieve higher *precision* and *recall* while reducing *FPR*.
- For OS and Docker, the use of *WorkloadN* to train classifiers produces better results in terms of both *precision* and *FPR*.

5.2.4 Algorithms Analysis

For this section's analysis, we focus upon the window size of BoSC and STIDE and upon the decision threshold for HMM. For this analysis, we filtered the results by training time, using `24H` for BoSC and STIDE and `2H` for HMM. In addition, each row of Table 5.5 consists of 900 entries.

Starting by OS results, we are able to observe a decrease in *precision* values (from `0.948` to `0.838`) for BoSC classifiers while the *recall* increases from `0.631` to `0.673` with the increase of the window size from `3` to `6`. Although there is an increase from `0.000` to `0.001` when the window size increases from `3` to `4`, the *FPR* remains unaltered for other window sizes. STIDE demonstrates a similar behaviour, increasing the *recall* value from `0.633` to `0.700` while *precision* decreases from `0.937` to `0.684` with window increase. *FPR* increases from `0.000` to `0.003`.

In connection with LXC, we are able to observe a similar behaviour for both BoSC and STIDE as well. The values of *precision* drops (from `0.878` to `0.552`, for BoSC and from `0.828` to `0.463`, for STIDE) whereas *recall* increases (from `0.853` to `0.979`, for BoSC and from `0.884` to `0.987`, for STIDE) as well as *FPR* (from `0.001` to `0.006`, for BoSC and from `0.002` to `0.010`, for STIDE).

While regarding Docker, we could observe a similar behaviour as well. For both BoSC and STIDE, we can observe an increase in *recall* and *FPR* results while *precision* drops with window growth. However, the case is portrayed differently for HMM. When the decision threshold increases to `100`, there is a significant increase in both *precision* (from `0.039` to

`0.767`) as well as *F-Measure* (from `0.074` to `0.854`) and in addition a major reduction in the *FPR* (from `0.679` to `0.008` when the decision threshold grows from `50` to `100`.

These observations reasons lie with the fact that an increase in window size causes a greater increase in the number of possible combinations, which as a consequence, decreases the stability of the profile defined by BoSC and STIDE classifiers. However, for HMM we denote a huge increaes and improvement in the results obtained with the exchange of the decision threshold from `50` to `100`. This threshold represents the logarithmic probability from which sequences start being considered as anomalous. Therefore, this means that the decision threshold `50` is still very low and most probably within the normal behaviour of the profile, which causes the huge impact of the raising of the decision threshold. Still, the behaviour of HMM is not regular since if the decision threshold is increased further, to `125` for instance, the results regress which demonstrates the sensitivity of the parameter.

Summary

- Both the window size and the decision threshold have great impact upon the results of HMM.
- BoSC and STIDE have a results improvement with the decrease of the window size
- Window size `3` and `4` produce the better results for BoSC and STIDE whereas decision threshold `100` and `125` are the most satisfactory for HMM

5.2.5 Generalisation Analysis

In this section, we step further into the examination of the impact of each training workload in the results produced. The results presented on Table 5.6 are filtered by training time, `24H`, for BoSC and STIDE, and `2H` for HMM, and by window size, for BoSC and STIDE, using window `3` and `4` whereas HMM uses decision thresholds of `100` and `125`.

Here, we aim to analyse the generalisation capacity granted to classifiers by each training workload type. For this, we cross-out the training and testing workload, analysing the results produced by classifiers trained with *WorkloadS* when testing *WorkloadN* datasets, and vice-versa, and assessing which performs better.

Focusing on OS deployment, it is possible to observe the improvement on results when classifiers are trained with the non-steady workload. Data shows better results in terms of *recall* and *precision* as well as lower *FPR*, for both BoSC and STIDE. The same observations can be made with Docker, while *FPR* drops, the value of *recall* and *precision* increases as a result of training classifiers with *WorkloadN*.

However, for LXC, there are a mixed behaviour. While BoSC classifiers behave differently, STIDE classifiers demonstrate the same attitude towards the training workload as OS and Docker classifiers. In this case, BoSC classifiers obtained better results when trained with *WorkloadS*, with higher *recall* and *precision* whereas STIDE produces superior results when classifiers are trained with the non-steady workload, in terms of *recall* and *precision*.

As expected, the majority of classifiers produce superior results when the applied training

Table 5.5: Analysis of algorithms window size and decision threshold for all classifiers for all platforms.

Platform	Algorithm	WS/ Dec. Thr.	Recall	Precision	F-Measure	FPR
Docker	STIDE	3	0.988	0.814	0.893	0.006
		4	0.990	0.786	0.876	0.007
		5	0.991	0.745	0.851	0.008
		6	0.992	0.682	0.809	0.011
	BoSC	3	0.987	0.821	0.896	0.005
		4	0.989	0.800	0.885	0.006
		5	0.990	0.782	0.874	0.007
		6	0.991	0.758	0.859	0.008
	HMM	50	0.991	0.039	0.074	0.679
		100	0.963	0.767	0.854	0.008
		125	0.765	0.745	0.755	0.007
		150	0.659	0.729	0.692	0.007
LXC	BoSC	3	0.853	0.878	0.865	0.001
		4	0.946	0.704	0.807	0.003
		5	0.966	0.607	0.746	0.005
		6	0.979	0.552	0.706	0.006
	STIDE	3	0.884	0.828	0.855	0.002
		4	0.972	0.635	0.769	0.005
		5	0.980	0.543	0.699	0.007
		6	0.987	0.463	0.631	0.010
OS	BoSC	3	0.631	0.948	0.758	0.000
		4	0.640	0.918	0.754	0.001
		5	0.652	0.882	0.749	0.001
		6	0.673	0.838	0.746	0.001
	STIDE	3	0.633	0.937	0.755	0.000
		4	0.641	0.880	0.742	0.001
		5	0.656	0.806	0.724	0.001
		6	0.700	0.684	0.692	0.003

WS: Window Size ; Dec. Thr.: Decision Threshold

workload is non-steady. This behaviour occurs as a consequence of the larger diversity of the datasets which are collected from the execution of the non-steady workload.

Summary

- All OS and Docker classifier as well as STIDE classifiers for LXC produce higher results in terms of *recall* and *precision*.
- BoSC classifiers trained with the steady workload present higher *recall* and *precision* when compared to BoSC classifiers trained with the non-steady workload.

Table 5.6: Analysis of training workload generalisation capacity for all platforms.

Platform	Train Workload	Test Workload	Algo	Recall	Prec.	F-Meas.	FPR
Docker	WorkloadS	WorkloadN	BoSC	0.985	0.729	0.838	0.008
			STIDE	0.986	0.720	0.832	0.009
			HMM	0.796	0.662	0.723	0.010
	WorkloadN	WorkloadS	BoSC	0.990	0.863	0.922	0.004
			STIDE	0.991	0.850	0.915	0.005
			HMM	0.928	0.834	0.878	0.005
LXC	WorkloadS	WorkloadN	BoSC	0.896	0.789	0.839	0.002
			STIDE	0.924	0.702	0.798	0.003
	WorkloadN	WorkloadS	BoSC	0.881	0.728	0.797	0.002
			STIDE	0.931	0.718	0.811	0.003
OS	WorkloadS	WorkloadN	BoSC	0.566	0.896	0.693	0.001
			STIDE	0.566	0.869	0.685	0.001
	WorkloadN	WorkloadS	BoSC	0.692	0.943	0.798	0.000
			STIDE	0.692	0.906	0.784	0.001

Algo: Algorithm ; Prec.: Precision ; F-Meas.: F-Measure

- As expected the majority of classifiers trained with *WorkloadN* produces better results in terms of *recall* and *precision*.

5.2.6 ROC and Expected Cost Analysis

For quite some time, the use of Receiver Operating Characteristic (ROC) curves has been well accepted, and, in most cases, enough to provide a clear visual comprehension of a classifier's performance. As explained in Section 2.2.2, ROC combines FPR and recall to create a curve containing all the different configurations for such classifier. However, as this selection manner does not take into account the characteristics of the environment, it may suffer from bias inherent to both recall and FPR. As a consequence, the expected cost analysis which uses not only recall and FPR as ROC curves but also the hostility of the operating environment as a way to try to mitigate the referred biases.

In this section, we present a ROC curves group for each platform, with the exception of Docker, for which we present two charts with ROC curves due to visual incompatibilities of ROC curves from BoSC and STIDE with HMM. Moreover, the results presented are filtered by train time, using `24H` for BoSC and STIDE, and `2H` for HMM, and the use of a window size of both `3` and `4`, and a decision threshold of either `100` or `125` for HMM.

Due to visual restrictions, for Docker the HMM results and the results from BoSC and STIDE were put into separate charts.

With regard to the results, for BoSC and STIDE, produced to show on Figure 5.6, it occurs a range in the values of FPR from `0.002` to `0.013` whereas recall ranges from `0.975` to `1.000`. In this case, unlike LXC and OS, the results produced are extremely similar among

different algorithm which use the same configuration.

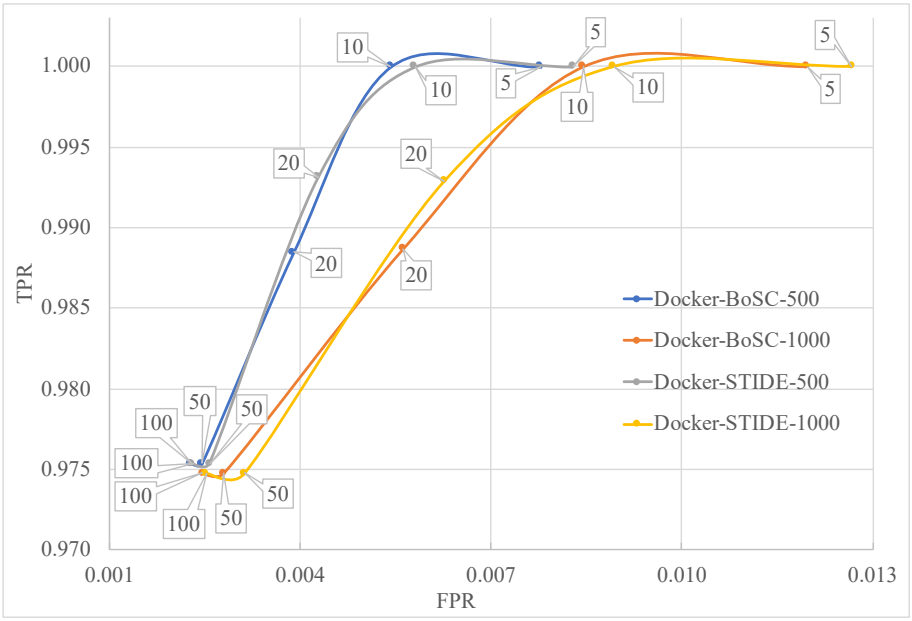


Figure 5.6: ROC curve for classifiers used for Docker deployment testing, for BoSC and STIDE.

In this case, we can observe better results for the BoSC algorithm with an epoch size of `500` due to the higher growth, achieving higher results of recall while maintaining FPR low.

The analysis of the set of ROC curves from Figure 5.7 allows to observe the range of results for FPR from `0.001` to `0.046` and the range of recall values from `0.790` to `0.907`. While the two minor epoch sizes produce a lower values of FPR, the use of HMM with epoch size of `5000`.

In connection with the results visible on the ROC curves for LXC containers, we can observe, in general terms, a range of `0.00` to `0.007` in terms of FPR, and a recall varying from `0.703` to `1.00`, without splitting the results by algorithm. Nonetheless, when observing the results individually, by algorithm and epoch size, it is clearly noted the rapid ascension of BoSC classifier with `500` of epoch size, which ranges from `0.703` with a detection threshold of `100` and achieves a `1.000` recall for a detection threshold of `5`. Although all epochs sizes with detection threshold of `5` achieve a recall value of `1.00`, they are distinguishable through the difference in the value of FPR, which gradually increase with the number of windows/sequences within an epoch and with the algorithms being used. The analysis of the ROC curves shows STIDE produces a higher value for FPR comparing to BoSC.

However, in general, the results for FPR are quite small, ranging from `0.000` to `0.007` from the results observed in Figure 5.8. Finally, we can based on ROC decide upon the best classifier and epoch size configuration for LXC containers, which is the use of BoSC with an epoch of size `500`.

Therefore, we present the set of ROC curves for the OS deployment in Figure 5.9, for BoSC and STIDE. The Figuredepicts an increase of recall with an increase of the FPR value. All four classifiers configurations demonstrate achieving lower FPR values when the

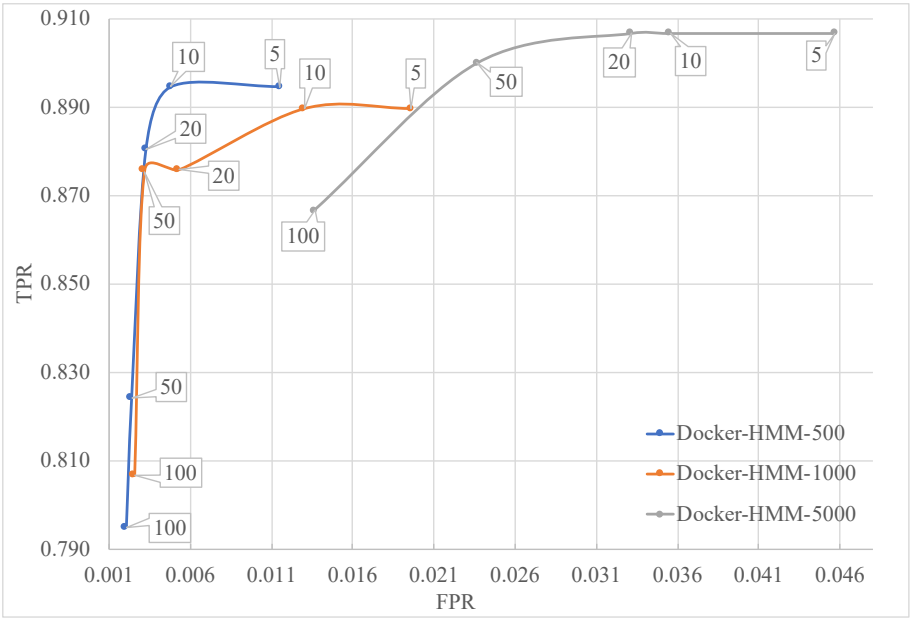


Figure 5.7: ROC curve for classifiers used for Docker deployment testing, for HMM.

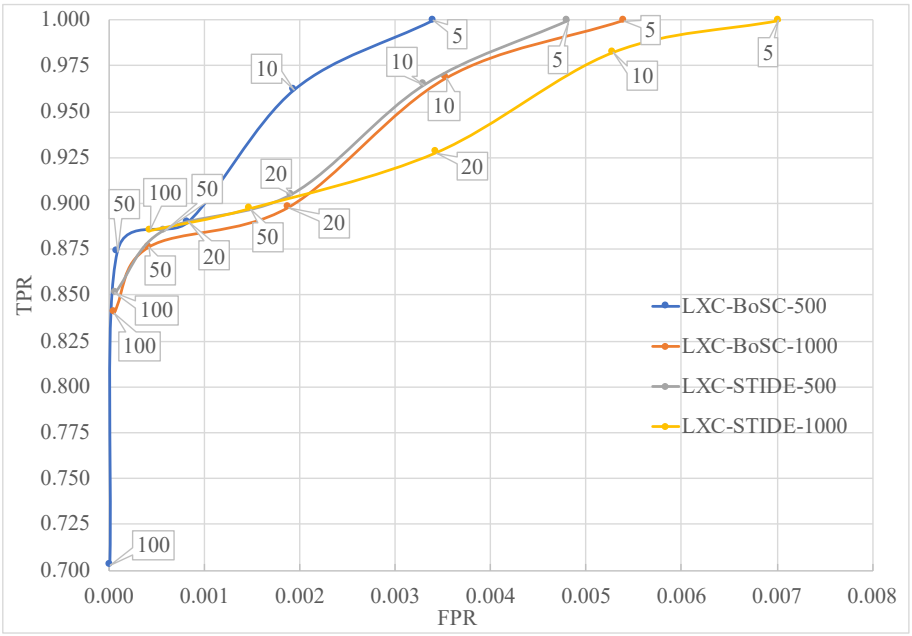


Figure 5.8: ROC curve for classifiers used for LXC deployment testing.

detection threshold value corresponds to the highest, that is `100`. A more in-depth analysis, clearly depicts the curve for BoSC using an epoch size of `500` as the one which grows faster and, globaly, results in a smaller FPR that the other for the same configuration despite achieving the same levels of recall.

For this curve, we are able to observe a range of `0.0001` to `0.0006` for FPR and a recall within the interval from `0.5927` to `0.6760` for BoSC with an epoch size of `500` whereas for the curve of STIDE using an epoch size of `1000` the decrease of the detection threshold value, provokes a larger increase on the FPR.

The FPR values reported for these configurations are quite small, ranging from `0` to `0.0016`

while recall ranges from 0.593 to 0.677. In this preliminary example we are able, through the analysis of results depicted, to select the best configuration of algorithm and epoch size, from which results the selection of BoSC with an epoch size of 500.

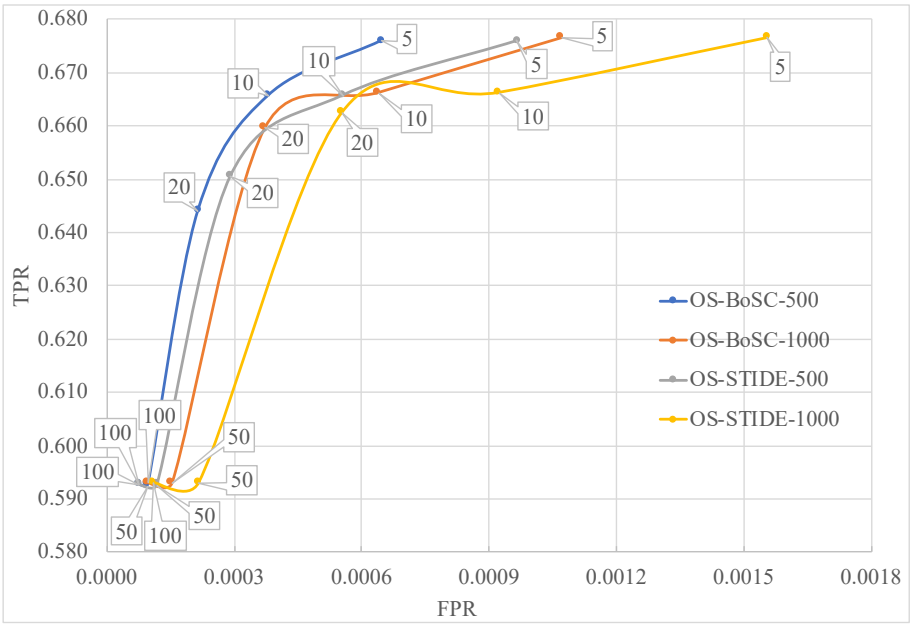


Figure 5.9: ROC curve for classifiers used for OS deployment testing.

In order to take into account the hostility of the operating environment, we also conducted an expected cost analysis as a way to achieve a specific value and proceed to compare the different classifiers based upon it. After selecting the best results for each combination of algorithms and epoch size, we condensed the results on Table 5.7 as a comparison manner for all the configurations defined.

The main aim of this comparison is to select the least costly configuration of algorithm, epoch size and detection threshold. Therefore, on Table 5.7 we can observe the expected cost for each configuration, and starting from the OS deployments cost, we can see that the cost for each option resides in the same region, ranging the values from 0.0305 to 0.0332. In this case, we are able to select the best performing configuration for BoSC and STIDE, with a epoch size of 500 and a detection threshold of 5.

Regarding LXC, we obtain costs ranging from 0.0034 to 0.0123. The best results for this case, are also with the configuration of 500 for epoch size and 5 for detection threshold, with expected costs of 0.0034 for BoSC and 0.0048 for STIDE.

In connection with Docker, we can observe that HMM results in higher costs than both BoSC and STIDE. While BoSC and STIDE produce costs ranging from 0.0053 to 0.0118, HMM results in costs in the region of 0.0318 to 0.0579. Nonetheless, all three algorithms produce the lowest expected costs for the configuration of epoch size of 500 and detection threshold of 10.

Table 5.7: Expected Cost Analysis for least costly configurations for each platform.

Platform	Algorithm	Epoch Size	Detection Threshold	P(I)	Recall	FPR	Expected Cost
Docker	BoSC	500	10	0.024	1.000	0.005	0.0053
		1000	20	0.024	0.989	0.006	0.0082
		5000	100	0.025	0.970	0.004	0.0112
	STIDE	500	10	0.024	1.000	0.006	0.0057
		1000	20	0.024	0.993	0.006	0.0078
		5000	100	0.025	0.970	0.004	0.0118
	HMM	500	10	0.026	0.895	0.005	0.0318
		1000	50	0.027	0.876	0.003	0.0369
		5000	50	0.035	0.900	0.024	0.0579
LXC	BoSC	500	5	0.008	1.000	0.003	0.0034
		1000	5	0.008	1.000	0.005	0.0054
		5000	100	0.008	0.876	0.001	0.0109
	STIDE	500	5	0.008	1.000	0.005	0.0048
		1000	10	0.008	0.982	0.005	0.0067
		5000	100	0.008	0.896	0.004	0.0123
OS	BoSC	500	5	0.009	0.677	0.001	0.0305
		1000	5	0.009	0.677	0.001	0.0309
		5000	20	0.009	0.667	0.001	0.0329
	STIDE	500	5	0.009	0.676	0.001	0.0308
		1000	5	0.009	0.677	0.002	0.0314
		5000	20	0.009	0.670	0.002	0.0332

Summary

- All results demonstrate that higher detection thresholds produce smaller recall and FPR values due to necessity of higher number of anomalous windows/sequences.
- All configuration with epoch size of `500` produce the least amount of expected cost.
- The detection thresholds which produce lower expected cost are `5`, for OS and LXC, and `10` for Docker.

5.2.7 Analysis of the Report Distribution

During this section, we aim to analyse the reports produced by the classifiers during the three different phases of the testing stage. As mentioned earlier, we segmented the testing stage into three phases: pre-attack phase (previous), attack phase (during) and post-attack phase (after) (see Figure 5.3 for details). The charts present in this section depict the distribution of the alarms percentage raised by the classifiers trained with both *workloadS* and *workloadN*.

Data presented in the charts was filtered, as before, by training time, `24H` for BoSC and STIDE and `2H` for HMM classifiers, moreover, we have also filtered through the window

size of 3 and 4 for BoSC and STIDE whereas for HMM we selected the decision threshold of 100 and 125. In addition, we crossed all the information regarding epoch size and detection threshold as well as algorithms and separated it by training workload, while the results from classifiers trained with *WorkloadS* are on left of the chart, the ones trained with *WorkloadN* are on the right.

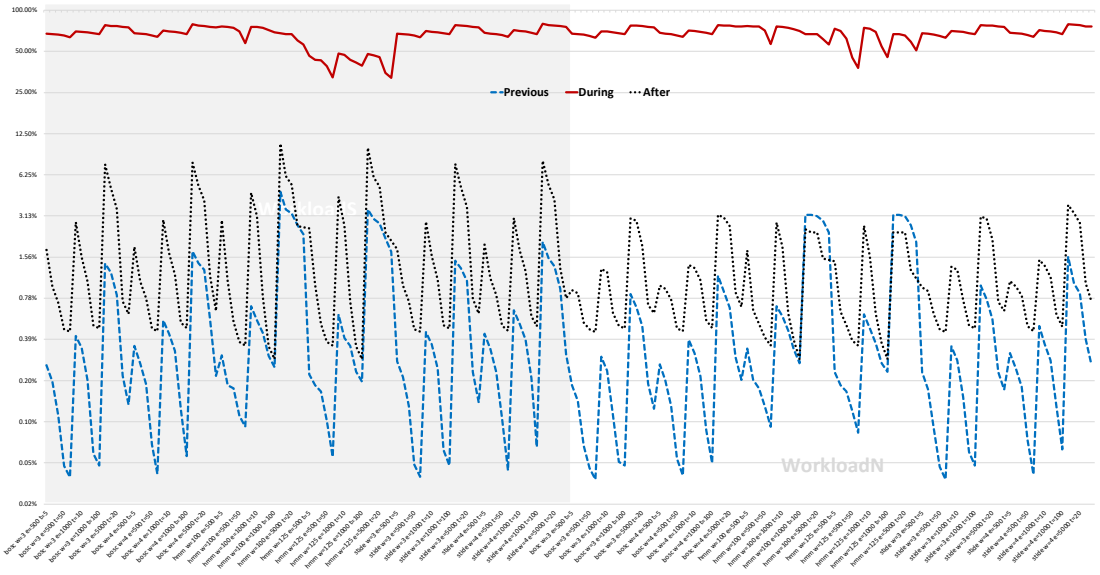


Figure 5.10: Distribution of the reports for Docker deployment according to the phase, with *WorkloadS* (steady workload) on the left and *WorkloadN* (non-steady workload) on the right.

Focusing on the results from the analysis of Docker reports, we can clearly note a wide gap between the curve of the percentage of reports during the attack phase when compared to the curve of both pre-attack phase and post-attack phase. While the results of the attack phase range from 32% to 79.55%, the other two phases range from 0% to 10.56%.

In addition, the analysis of both halfs of the chart, shows without doubt a decrease in the reports for classifiers trained with the non-steady workload. Classifiers trained with the *WorkloadS* produce a range of reports with a percentage ranging from 0% to 10.56% whereas classifiers fed with data from the non-steady workload for training resulted in a percentage ranging from 0% to 3.77%. Thus it is possible to conclude that, for Docker, *WorkloadN* produces more precise results.

Regarding the report issued for LXC containers, we can observe an almost clear distinction between the curve for the attack phase and the pre-attack and post-attack phases, with the exception of 4 points which produce an extremely low percentage of reports. The configuration of these points holds on an epoch size of 500 and a detection threshold of 100. Once more, this very strict configuration limits the number of anomaly reports raised due to the high percentage of anomaly windows/sequences required. In these cases, the percentage of reports drops to a value lower than 0.2%, nonetheless this percentage drop is also verified for the other two phases of detection, where the percentage of reports drops to nearly 0% in these cases. With the exception of these outliers, the remaining results do not exceed the value of 2.4% for pre-attack and post-attack phases whereas during the attack phase the percentage of reports is, in general, at least, 3.1% and producing at most

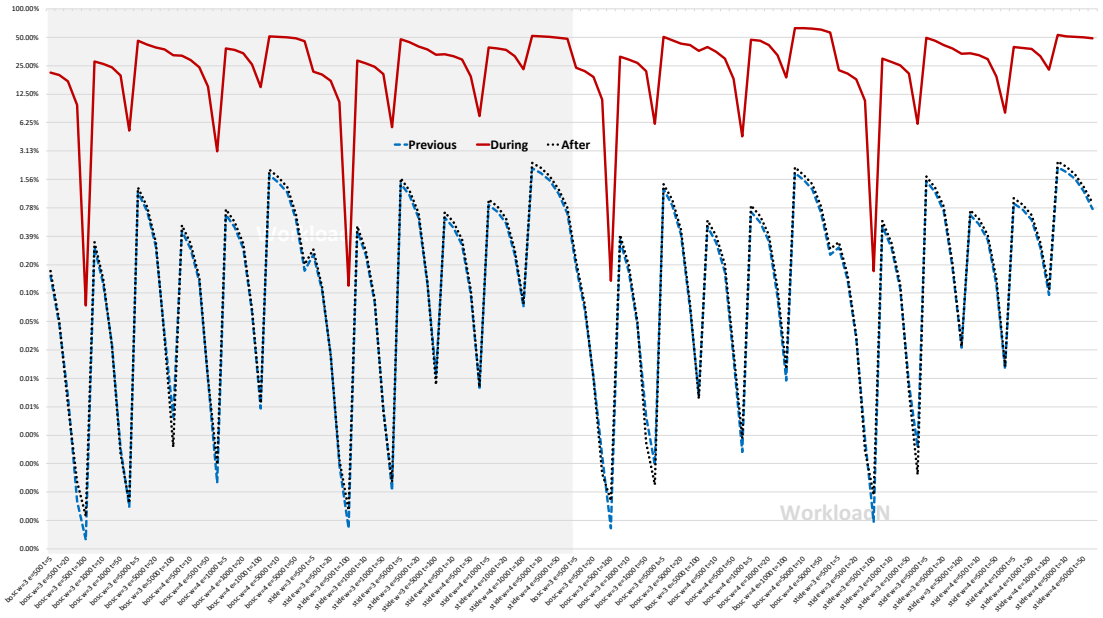


Figure 5.11: Distribution of the reports for LXC deployment according to the phase, with *WorkloadS* (steady workload) on the left and *WorkloadN* (non-steady workload) on the right.

a percentage of 63% of reports.

In addition, we can observe slight increases for cases such as window size of `4` for BoSC and an epoch size of *5000* with a detection threshold of `5`. With this configuration is possible to note an increase in the percentage of reports from 51.16% (with *WorkloadS*) to 62.57% (with *WorkloadN*).

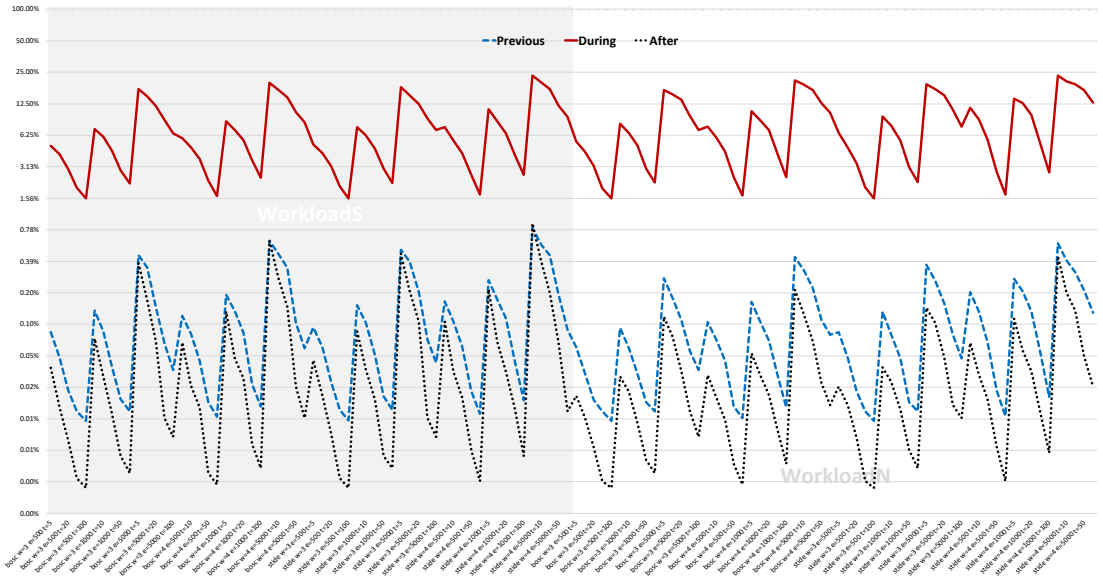


Figure 5.12: Distribution of the reports for OS deployment according to the phase, with *WorkloadS* (steady workload) on the left and *WorkloadN* (non-steady workload) on the right.

With regard to the results produced for the OS deployment, we are able to observe that during the attack phase there are higher percentage of reports, which was obviously expected. In this phase, the values registered range from 1.56% to nearly 25%. Although these percentages are not as high as expected, they are clearly distinguishable from the other two phases, which report rates are below 1%.

In connection with the attacking phase curve, we observe the lowest percentage at points where the epoch size is `500` and the detection threshold `100`. As a consequence, for an anomaly report to be issued with such configuration, it is required a level of anomaly higher than 20% within a given epoch. This configuration is actually the more strict, in terms of number of windows/sequences anomalous within an epoch. Therefore, this fact combined with possibly traces not very distinct results in lower report rate.

On the other hand, the highest report rate registered occurs with a configuration utilising an epoch size of `5000` and a detection threshold of `5`. The main reason for this is the opposite to the one stated before. In this case, we have the most liberal configuration where it is only required a 0.1% of anomalous windows within a certain epoch to raise an anomaly alert.

Despite these rates to produce an alert, the curves for pre-attack and post-attack phases remain low, indicating the isolated act of the attack.

Moreover, despite some slight improvements in the reports percentage on some occasions, such as STIDE with window of size `4`, epoch size of `500` and detection threshold of `5`, all curves seem to have the same behaviour independently of the training workload under utilisation.

Summary

- The percentage of reports is higher during the attack phase for every platform
- It is possible to note, on some occasions for all platforms, an increase in the percentage of reports during the attack phase and a decrease during the other two phases
- Docker has the widest gap between the attack phase and the other ones and produces reports very focused on the attack phase, while rarely reporting anomalies outside of this scope

5.2.8 Analysis of Results per Exploit

In this section, an exploit-focused analysis is performed in order to comprehend the type of attacks which are more easily picked up by the classifiers. In addition, the results presented in Table 5.8 were subjected to filtering by training time and window size and decision threshold.

For CVE-2016-6662, classifiers seem to perform poorly in detecting the attack. Regarding LXC and Docker, the almost no classifier surpasses the 50% barrier in both *recall* and *precision*, with the exception of BoSC and STIDE which registered `0.516` and `0.550` for *recall* for Docker. In addition, HMM actually reported 0% in both *recall* and *precision* for

the exploitation of this vulnerability. However, in the case of the OS deployment, we can observe an average performance given that *recall* values are `0.558` for BoSC and `0.572` for STIDE whereas the value of *precision* was `0.726` and `0.647` for BoSC and STIDE, respectively.

Table 5.8: Analysis of results with focus on the Exploits utilised.

Platform	Exploit No.	Algorithm	Recall	Prec.	F-Meas.	FPR
Docker	CVE-2016-6662	BoSC	0.516	0.324	0.398	0.003
		STIDE	0.550	0.308	0.395	0.003
		HMM	0.000	0.000	0.000	0.004
	CVE-2012-5611	BoSC	1.000	0.680	0.810	0.013
		STIDE	1.000	0.669	0.802	0.013
		HMM	1.000	0.640	0.780	0.017
	CVE-2013-1861	BoSC	1.000	0.789	0.882	0.011
		STIDE	1.000	0.779	0.876	0.012
		HMM	1.000	0.771	0.871	0.013
	CVE-2012-5627	BoSC	1.000	0.954	0.977	0.002
		STIDE	1.000	0.952	0.975	0.002
		HMM	0.709	0.953	0.813	0.002
	CVE-2016-6663	BoSC	1.000	0.918	0.957	0.001
		STIDE	1.000	0.912	0.954	0.002
		HMM	0.983	0.831	0.901	0.004
LXC	CVE-2016-6662	BoSC	0.386	0.322	0.351	0.002
		STIDE	0.440	0.256	0.324	0.003
	CVE-2012-5611	BoSC	1.000	0.838	0.912	0.002
		STIDE	1.000	0.814	0.898	0.003
	CVE-2013-1861	BoSC	1.000	0.451	0.621	0.002
		STIDE	1.000	0.341	0.509	0.003
	CVE-2012-5627	BoSC	0.608	0.571	0.589	0.002
		STIDE	0.695	0.491	0.576	0.003
	CVE-2016-6663	BoSC	0.974	0.912	0.942	0.002
		STIDE	0.993	0.870	0.927	0.004
OS	CVE-2016-6662	BoSC	0.558	0.726	0.631	0.001
		STIDE	0.572	0.647	0.607	0.001
	CVE-2012-5611	BoSC	1.000	0.938	0.968	0.001
		STIDE	1.000	0.917	0.957	0.001
	CVE-2013-1861	BoSC	1.000	0.973	0.986	0.000
		STIDE	1.000	0.961	0.980	0.001
	CVE-2012-5627	BoSC	0.006	0.275	0.012	0.000
		STIDE	0.006	0.216	0.012	0.000
	CVE-2016-6663	BoSC	0.999	0.967	0.982	0.000
		STIDE	0.999	0.959	0.979	0.000

Furthermore, CVE-2012-5611 and CVE-2013-1861 demonstrate a similar performance for all classifiers from every platform, with high *recall* values and *precision* and low *FPR*. Nonetheless, there is an exception with regard to the classifiers of BoSC and STIDE for

LXC, which demonstrate a decrease value for *precision*, 0,451 for BoSC and 0.341 for STIDE.

While Docker classifiers easily and with high *precision* detect both CVE-2012-5627 and CVE-2016-6663, LXC and OS classifiers struggle to detect the exploitation of CVE-2012-5627, which is left undetected by the classifiers of the OS deployment, where it was registered a values of 0.006 for *recall* and around 20% for *precision*. Also for CVE-2012-5627, LXC classifiers produce a *recall* of 0.608 for BoSC and 0.695 for STIDE whereas for *precision* produced 0.571 and 0.491, respectively.

The lack of capacity in detecting the exploit for CVE-2016-6662 resides in the short attack span and the reduced number of modifications. Nonetheless, at the OS deployment the detection is pretty successful.

Summary

- The exploitation of CVE-2012-5611, CVE-2013-1861 and CVE-2016-6663 are easily detected fully, producing *recall* values in the region of 100%, however in some cases *precision* is not as high due to attack consequences which are deemed anomalous but are outside of the attack span.
- HMM classifier does not detect the CVE-2016-6662 exploitation.
- CVE-2012-5627 produces very diverse results, although for Docker the attack is clearly detected, for LXC the results are in the region of 50-60% for *recall* and *precision* while for OS the attack is not detected.

5.2.9 Analysis of the Best Cases for each Platform

For this final section, we selected the best configuration for all platforms, in order to compile in a table the best results for each one. All results presented in the following tables resulted from a 24H, for BoSC and STIDE and a 2H, for HMM, training period.

In connection to Docker, we are able to observe a nearly constant high value for recall across all configurations, while precision may suffer on some occasions. The values registered for recall range from 0.821 to 1.000, whereas precision remains in the interval of 0.549 to 0.838 and FPR ranges from 0.005 to 0.020. For this platform the analysis of Table 5.9 permits to draw the conclusion that the most efficient, in terms of F-Measure, is the BoSC algorithm with a window of size 3, an epoch size of 500 and a detection threshold of 10.

Focusing on LXC, we can observe high levels of recall for every entry in Table 5.10, having a range of 0.926 to 1.000, while precision values range from 0.484 to 0.928 and FPR starting at 0.001 to 0.009. In this case, the highest value for F-Measure is produced with a BoSC classifier using a window of size 3, an epoch size of 500 and a detection threshold of 10.

With regard to OS, the analysis of Table 5.11 allows to conclude the incapacity of the algorithms to overcome the 0.687 barrier for recall values. Nevertheless, these classifiers achieve very high precision values for the most of configurations, with 0.957 as the highest

Table 5.9: Docker deployment best case with 24H for BoSC and STIDE and 2H for HMM of Training Time.

Algorithm	**WS/ Dec. Thr.**	**Epoch Size**	**Detection Threshold**	**Recall**	**Prec.**	**F-Meas.**	**FPR**
BoSC	3	500	5	1.000	0.770	0.870	0.007
			10	1.000	0.830	0.907	0.005
		1000	5	1.000	0.685	0.813	0.011
			10	1.000	0.756	0.861	0.008
	4	500	5	1.000	0.751	0.858	0.008
			10	1.000	0.811	0.896	0.006
		1000	5	1.000	0.664	0.798	0.013
			10	1.000	0.736	0.848	0.009
STIDE	3	500	5	1.000	0.763	0.866	0.008
			10	1.000	0.823	0.903	0.005
		1000	5	1.000	0.677	0.808	0.012
			10	1.000	0.749	0.857	0.008
	4	500	5	1.000	0.735	0.848	0.009
			10	1.000	0.797	0.887	0.006
		1000	5	1.000	0.647	0.786	0.014
			10	1.000	0.722	0.839	0.010
HMM	100	500	5	0.965	0.670	0.791	0.013
			10	0.965	0.838	0.897	0.005
		1000	5	0.959	0.569	0.714	0.020
			10	0.959	0.654	0.778	0.014
	125	500	5	0.824	0.675	0.742	0.010
			10	0.824	0.822	0.823	0.005
		1000	5	0.821	0.549	0.658	0.019
			10	0.821	0.659	0.731	0.012

one registered for window size of `3`, epoch size of `500` and a detection threshold of `10`, where it also resulted in a `0.000` for FPR.

Table 5.10: LXC deployment best case with 24H of Training Time.

Algorithm	Window Size	Epoch Size	Detection Threshold	Recall	Prec.	F-Meas.	FPR
BoSC	3	500	5	0.999	0.820	0.901	0.002
			10	0.926	0.928	0.927	0.001
		1000	5	0.999	0.701	0.824	0.003
			10	0.938	0.832	0.882	0.002
	4	500	5	1.000	0.603	0.753	0.005
			10	1.000	0.697	0.822	0.003
		1000	5	1.000	0.510	0.676	0.007
			10	1.000	0.580	0.734	0.006
STIDE	3	500	5	0.999	0.745	0.854	0.003
			10	0.929	0.862	0.894	0.001
		1000	5	0.999	0.629	0.772	0.005
			10	0.964	0.743	0.839	0.003
	4	500	5	1.000	0.558	0.716	0.007
			10	1.000	0.613	0.760	0.005
		1000	5	1.000	0.484	0.652	0.009
			10	1.000	0.523	0.687	0.008

Table 5.11: OS deployment best case with 24H of Training Time.

Algorithm	Window Size	Epoch Size	Detection Threshold	Recall	Prec.	F-Meas.	FPR
BoSC	3	500	5	0.666	0.925	0.774	0.000
			10	0.666	0.957	0.785	0.000
		1000	5	0.666	0.884	0.760	0.001
			10	0.666	0.927	0.775	0.000
	4	500	5	0.686	0.889	0.775	0.001
			10	0.666	0.927	0.775	0.000
		1000	5	0.687	0.829	0.751	0.001
			10	0.666	0.888	0.761	0.001
STIDE	3	500	5	0.666	0.911	0.769	0.001
			10	0.666	0.946	0.781	0.000
		1000	5	0.666	0.861	0.751	0.001
			10	0.666	0.910	0.769	0.001
	4	500	5	0.686	0.828	0.750	0.001
			10	0.666	0.891	0.762	0.001
		1000	5	0.687	0.752	0.718	0.002
			10	0.666	0.834	0.741	0.001

Chapter 6

Conclusions and Future Work

The increasing usage of containers for cloud services provides high scalability and the capability to efficiently manage resources according to current demand. In addition, the possibility to share physical resources for multiple tenants at a lower level and closer to the Operating System (OS) contributes to performance improvements. The portability granted by this technology allows building faster and without taking into account the worries of underlying technologies, such as the OS. These features encourage its adoption since we live in a rapidly changing world and the time-to-market is extremely important to companies and the services they provide.

However, the security concerns widen as the structure is shared by multiple actors and some might have malicious intentions toward its tenant neighbours. Thus, in addition to the traditional security concerns, the provider must also be concerned in assuring the identification of malicious containers and the security of non-malicious ones. Intrusion detection is a technique with proven results and we argue it is a must have for container-based multi-tenant environments, where the detection of attacks from and against containers within the infrastructure is of utmost importance for service providers as well as their customers.

In this work we proposed an experimental methodology to rigorously evaluate the effectiveness of intrusion detection algorithms in container-based systems. For this, we adopted the best practices from system's performance evaluation such as representative workloads and state of the art concepts of attack injection. We adapted widely used metrics to be used, defined representative setups and automated the test execution and results analysis tasks. This methodology guided the experiments performed in this work.

The results obtained during our experiments provide evidences of the applicability and effectiveness of state-of-the-art anomaly based intrusion detection algorithms for containerised deployments. The attacks were clearly detected and the results also demonstrate low false positives reports. The algorithms BoSC and STIDE demonstrated to be effective with both high recall and high precision while producing low false positives.

HMM was also evaluated, but we only obtained results for Docker containers and in this case it produced as well high recall values, although not has high as for BoSC and STIDE, and low false positives, for some configurations as low as BoSC and STIDE. However, the scalability limitations observed in the algorithm most likely limit its adoption in practice.

We also observed that, as expected, the configuration of the intrusion detection algorithm impacts its performance. This is quite common, as the careful tuning of IDSes is a frequent requirement in other domains. In our experiments, windows of size 3 and 4 performed better, combined with epochs of length 500 and 1000 with thresholds of 5 and 10. Nevertheless, good results were observed for most of the configurations, evidencing that this is no product of overfitting effects. In fact, our analysis of generalisation shows that even when using workloads with different operation profiles, the algorithms achieved good results.

It was also shown that the increase in the training time of classifiers produced better results, for every algorithms used in the experimental campaign. Although the training time, for STIDE and BoSC, of `24H` produces the most precise results, after `12H` of training the classifiers are already capable of producing acceptable results. This comes into line with the results of our preliminary study to the definition of stable profiles where, we achieved steady-state for the learning procedure under `10H`, in some occasions.

In general, the results observed were more satisfactory for container deployments when compared to the traditional OS-level deployment. A more complete study would be necessary to conclude that intrusion detection performs better in containers than in more traditional setups, such as virtual machines, but this is not one of our goals. Our experiments were sufficient to understand that while in containers it is quite easy to define the monitoring surface, and can be done in a highly portable way, in a traditional machine/OS we need the additional concern of defining this surface, and its incorrect definition may lead to unsatisfactory results.

Future Work

Future work includes the extension of the evaluation presented in this work by adding more representative and more complex systems, in order to diversify their range and prove the generalisation of the technique. In addition, the inclusion of new vulnerabilities and exploits would also be a great inclusion as a way to increase the representativeness of our results.

It also includes the extension of the evaluation to other state-of-the-art intrusion detection algorithms aiming to widen the range of techniques whose applicability to containerised systems have been studied. Among these algorithms, we can find Naïve Bayes (NB), One-Class Support Vector Machines (OCSVM) or K-Nearest Neighbour (KNN) due to its previous usage in this field.

Furthermore, this work may lead to the research of new security counter-measures. For instance, intrusion tolerance and reaction mechanisms would be a great addition to these environments. In fact, the containers environment has the necessary characteristics to instantiate techniques of intrusion tolerance based on replication and intrusion detection: these would take advantage of the rapid and inexpensive instantiation and reusable characteristics of containers.

Intrusion detection in compositions of containers is also an area that we plan to explore. The monitoring procedure applied to groups of containers whose operation is connected may result in the capability of detecting attacks more efficiently due to cross-referencing information collected from the different related sources.

References

[1] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *Computer Security Division, Information Technology Laboratory, National*, page 7, September 2011.

[2] Murugiah Souppaya, John Morello, and Karen Scarfone. Application container security guide. Technical Report NIST SP 800-190, National Institute of Standards and Technology, Gaithersburg, MD, September 2017.

[3] K A Scarfone, M P Souppaya, and P Hoffman. Guide to security for full virtualization technologies. Technical Report NIST SP 800-125, National Institute of Standards and Technology, Gaithersburg, MD, 2011.

[4] Michael Kerrisk. cgroups - Linux control groups. `http://man7.org/linux/man-pages/man7/cgroups.7.html`, October 2018. [Accessed: 2019-01-14].

[5] Michael Kerrisk. namespaces - overview of Linux namespaces. `http://man7.org/linux/man-pages/man7/namespaces.7.html`, October 2018. [Accessed: 2019-01-14].

[6] datadoghq. 8 surprising facts about real docker adoption. `https://www.datadoghq.com/docker-adoption/`, June 2018. [Accessed: 2019-01-02].

[7] Docker. `https://www.docker.com`. Accessed: 2019-04-10.

[8] Nick Antonopoulos. *Cloud computing: principles, systems and applications*. Springer Berlin Heidelberg, New York, NY, 2017.

[9] Marcello Cinque, Raffaele Della Corte, Antonio Eliso, and Antonio Pecchia. RT-CASEs: Container-Based Virtualization for Temporally Separated Mixed-Criticality Task Sets. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:22, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[10] Dario Sabella, Vadim Sukhomlinov, Linh Trang, Sami Kekki, Pietro Paglierani, Ralf Rossbach, Xinhui Li, Yonggang Fang, Dan Druta, Fabio Giust, Luca Cominardi, Walter Featherstone, Bob Pike, and Shlomi Hadad. Developing Software for Multi-Access Edge Computing. *ETSI White Paper No. 20*, page 38, February 2019.

[11] Aleksandar Milenkoski. *Evaluation of Intrusion Detection Systems in Virtualized Environments*. PhD thesis, Fakultät für Mathematik und Informatik, 2016.

[12] Yasir Mehmood, Muhammad Awais Shibli, Umme Habiba, and Rahat Masood. Intrusion Detection System in Cloud Computing: Challenges and opportunities. pages 59–66. IEEE, December 2013.

[13] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 120–128. IEEE Computer Society Press, 1996.

[14] Theuns Verwoerd and Ray Hunt. Intrusion detection techniques and approaches. *Computer Communications*, 25(15):1356–1365, sep 2002.

[15] Steven R Snapp, James Brentano, Gihan V Dias, Terrance L Goan, L Todd Heberlein, Che-Lin Ho, Karl N Levitt, Biswanath Mukherjee, Stephen E Smaha, Tim Grance, Daniel M Teal, and Doug Mansur. DIDS (Distributed Intrusion Detection System) - Motivation, Architecture, and An Early Prototype. In *Proceedings of the 14th national computer security conference*, volume 1, pages 167–176, Washington, DC, 1991.

[16] Amr S. Abed, Charles Clancy, and David S. Levy. Intrusion Detection System for Applications using Linux Containers. *arXiv:1611.03056 [cs]*, 9331:123–135, 2015.

[17] Siddharth Srinivasan, Akshay Kumar, Manik Mahajan, Dinkar Sitaram, and Sanchika Gupta. Probabilistic Real-Time Intrusion Detection System for Docker Containers. In *Security in Computing and Communications*, pages 336–347. Springer Singapore, 2019.

[18] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. Evaluating computer intrusion detection systems. *ACM Computing Surveys*, 48(1):1–41, sep 2015.

[19] Aleksandar Milenkoski, Bryan D. Payne, Nuno Antunes, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Matthias Luft. Evaluation of intrusion detection systems in virtualized environments using attack injection. In *International Symposium on Recent Advances in Intrusion Detection*, pages 471–492. Springer, 2015.

[20] Joao Antunes, Nuno Neves, Miguel Correia, Paulo Verissimo, and Rui Neves. Vulnerability Discovery with Attack Injection. *IEEE Transactions on Software Engineering*, 36(3):357–370, May 2010.

[21] Jose Fonseca, Marco Vieira, and Henrique Madeira. Evaluation of Web Security Mechanisms Using Vulnerability & Attack Injection. *IEEE Transactions on Dependable and Secure Computing*, 11(5):440–453, September 2014.

[22] Guy Bruneau. The History and Evolution of Intrusion Detection. page 8, 2001.

[23] Rebecca Bace and Peter Mell. NIST Special Publication on Intrusion Detection Systems. NIST Pubs 800-31, NIST, November 2001.

[24] Md Shariful Islam, Korosh Koochekian Sabor, Abdelaziz Trabelsi, Wahab Hamou-Lhadj, and Luay Alawneh. Masked: A mapreduce solution for the kappa-pruned ensemble-based anomaly detection system. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 25–34. IEEE, 2018.

[25] Wael Khreich, Syed Shariyar Murtaza, Abdelwahab Hamou-Lhadj, and Chamseddine Talhi. Combining heterogeneous anomaly detectors for improved software security. *Journal of Systems and Software*, 137:415–429, March 2018.

[26] Chirag N. Modi and Dhiren Patel. A novel hybrid-network intrusion detection system (H-NIDS) in cloud computing. In *2013 IEEE Symposium on Computational Intelligence in Cyber Security (CICS)*, pages 23–30, Singapore, Singapore, April 2013. IEEE.

[27] K A Scarfone and P M Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). Technical Report NIST SP 800-94, National Institute of Standards and Technology, Gaithersburg, MD, 2007.

[28] Anup K Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning Program Behavior Profiles for Intrusion Detection. *Proceedings of the Workshop on Intrusion Detection and Network Monitoring*, page 13, April 1999.

[29] Dae-Ki Kang, D. Fuller, and V. Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Proceedings from the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 2005.*, pages 118–125, West Point, NY, USA, 2005. IEEE.

[30] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344)*. IEEE Comput. Soc, 1999.

[31] Yihua Liao and V.Rao Vemuri. Use of K-Nearest Neighbor classifier for intrusion detection. *Computers & Security*, 21(5):439–448, October 2002.

[32] Chih-Fong Tsai, Yu-Feng Hsu, Chia-Ying Lin, and Wei-Yang Lin. Intrusion detection by machine learning: A review. *Expert Systems with Applications*, 36(10):11994–12000, December 2009.

[33] Wun-Hwa Chen, Sheng-Hsun Hsu, and Hwang-Pin Shen. Application of SVM and ANN for intrusion detection. *Computers & Operations Research*, 32(10):2617–2634, October 2005.

[34] Barnaby Stewart, Luis Rosa, Leandros A. Maglaras, Tiago J. Cruz, Mohamed Amine Ferrag, Paulo Simoes, and Helge Janicke. A Novel Intrusion Detection Mechanism for SCADA systems which Automatically Adapts to Network Topology Changes. *EAI Endorsed Transactions on Industrial Networks and Intelligent Systems*, 4(10):152155, February 2017.

[35] John McHugh. Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. *ACM Transactions on Information and System Security*, 3(4):262–294, November 2000.

[36] Robin Sommer and Vern Paxson. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, Oakland, CA, USA, 2010. IEEE.

[37] Yuxin Meng and Wenjuan Li. Adaptive Character Frequency-Based Exclusive Signature Matching Scheme in Distributed Intrusion Detection Environment. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 223–230, Liverpool, United Kingdom, June 2012. IEEE.

[38] Nuno Antunes and Marco Vieira. On the Metrics for Benchmarking Vulnerability Detection Tools. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 505–516, Rio de Janeiro, Brazil, June 2015. IEEE.

[39] David Martin Powers. Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, December 2011.

[40] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, February 2013.

[41] J.E. Gaffney and J.W. Ulvila. Evaluation of intrusion detectors: a decision theory approach. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, pages 50–61, Oakland, CA, USA, 2001. IEEE Comput. Soc.

[42] Michael Kerrisk. chroot - change root directory. `http://man7.org/linux/man-pages/man2/chroot.2.html`, September 2017. [Accessed: 2019-02-23].

[43] Matteo Riondato. Chapter 14. Jails. `https://www.freebsd.org/doc/handbook/jails.html`. [Accessed: 2019-02-23].

[44] Gabor Nagy. Operating System Containers vs. Application Containers. `https://blog.risingstack.com/operating-system-containers-vs-application-containers/`, May 2015. [Accessed: 2019-02-23].

[45] Linux Containers. `https://wiki.archlinux.org/index.php/Linux_Containers`, February 2019. [Accessed: 2019-03-01].

[46] Sysdig, Inc. sysdig. `https://sysdig.com/`, 2019. [Accessed: 2019-01-24].

[47] Shiv Dhar. Sysdig Overview. `https://github.com/draios/sysdig/wiki/Sysdig-Overview`, December 2017. [Accessed: 2019-01-24].

[48] Sysdig vs dtrace vs strace: A technical discussion. `https://sysdig.com/blog/sysdig-vs-dtrace-vs-strace-a-technical-discussion/`. Accessed: 2019-07-23.

[49] Michael Kerrisk. strace - trace system calls and signals. `http://man7.org/linux/man-pages/man1/strace.1.html`, October 2018. [Accessed: 2019-01-24].

[50] Google, inc. cAdvisor. `https://github.com/google/cadvisor`, August 2018. [Accessed: 2019-01-24].

[51] Massimiliano Mattetti, Alexandra Shulman-Peleg, Yair Allouche, Antonio Corradi, Shlomi Dolev, and Luca Foschini. Securing the infrastructure and the workloads of linux containers. In *2015 IEEE Conference on Communications and Network Security (CNS)*, pages 559–567, Florence, Italy, September 2015. IEEE.

[52] Nilton Bila, Paolo Dettori, Ali Kanso, Yuji Watanabe, and Alaa Youssef. Leveraging the Serverless Architecture for Securing Linux Containers. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 401–404, Atlanta, GA, USA, June 2017. IEEE.

[53] A. A. Mohallel, J. M. Bass, and A. Dehghantaha. Experimenting with docker: Linux container and base os attack surfaces. In *2016 International Conference on Information Society (i-Society)*, pages 17–21, Oct 2016.

[54] A. Martin, S. Raponi, T. Combe, and R. Di Pietro. Docker ecosystem – vulnerability analysis. *Computer Communications*, 122:30 – 43, 2018.

[55] Zhiqiang Jian and Long Chen. A defense method against docker escape attack. In *Proceedings of the 2017 International Conference on Cryptography, Security and Privacy*, pages 142–146. ACM, 2017.

[56] Xing Gao, Zhongshu Gu, Mehmet Kayaalp, Dimitrios Pendarakis, and Haining Wang. ContainerLeaks: Emerging Security Threats of Information Leakages in Container Clouds. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 237–248, Denver, CO, USA, June 2017. IEEE.

[57] A. Duarte and N. Antunes. An Empirical Study of Docker Vulnerabilities and of Static Code Analysis Applicability. In *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, pages 27–36, 2018.

[58] Tpc-c benchmark. `http://www.tpc.org/tpcc/`. Accessed: 2019-04-10.

[59] Mysql (linux) - stack buffer overrun (poc). `https://www.exploit-db.com/exploits/23075`. Accessed: 2019-04-10.

[60] Oracle mysql / mariadb - insecure salt generation security bypass. `https://www.exploit-db.com/exploits/38109`. Accessed: 2019-04-10.

[61] Mysql / mariadb - geometry query denial of service. `https://www.exploit-db.com/exploits/38392`. Accessed: 2019-04-10.

[62] Mysql / mariadb / perconadb 5.5.52 / 5.6.33 / 5.7.15 - code execution / privilege escalation. https://0day.today/exploit/24786.

[63] Mysql / mariadb / perconadb 5.5.x/5.6.x/5.7.x - 'mysql' system user privilege escalation / race condition. `https://www.exploit-db.com/exploits/40678`. Accessed: 2019-04-10.

[64] Vulnerability details : Cve-2012-5611. `https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2012-5611`. Accessed: 2019-04-10.

[65] Vulnerability details : Cve-2012-5627. `https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2012-5627`. Accessed: 2019-04-10.

[66] Vulnerability details : Cve-2013-1861. `https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2013-1861`. Accessed: 2019-04-10.

[67] Vulnerability details : Cve-2016-6662. `https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2016-6662`. Accessed: 2019-04-10.

[68] Vulnerability details : Cve-2016-6663. `https://www.cvedetails.com/cve-details.php?t=1&cve_id=CVE-2016-6663`. Accessed: 2019-04-10.

[69] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in Cloud. *Journal of Network and Computer Applications*, 36(1):42–57, January 2013.

Appendices

Appendix A

Complete Experimental Results for All Platforms

Below we list the complete results obtained from our experiments. To make their use and analysis easier, we provide them available online, in the form of a SQLite database:

– `https://github.com/jeflora/containers-ids-evaluation`.

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	30min	HMM	50	500	5	1.000	0.028	0.054	0.918
Docker	1	HMM	50	500	5	1.000	0.028	0.054	0.919
Docker	2	HMM	50	500	5	1.000	0.028	0.054	0.927
Docker	30min	HMM	50	500	10	1.000	0.037	0.072	0.678
Docker	1	HMM	50	500	10	1.000	0.038	0.072	0.678
Docker	2	HMM	50	500	10	1.000	0.037	0.071	0.697
Docker	30min	HMM	50	500	20	0.987	0.048	0.092	0.516
Docker	1	HMM	50	500	20	0.987	0.048	0.092	0.516
Docker	2	HMM	50	500	20	0.987	0.048	0.091	0.518
Docker	30min	HMM	50	500	50	0.983	0.049	0.093	0.505
Docker	1	HMM	50	500	50	0.983	0.049	0.093	0.505
Docker	2	HMM	50	500	50	0.983	0.049	0.093	0.505
Docker	30min	HMM	50	500	100	0.983	0.049	0.094	0.503
Docker	1	HMM	50	500	100	0.983	0.049	0.094	0.503
Docker	2	HMM	50	500	100	0.983	0.049	0.094	0.503
Docker	30min	HMM	50	1000	5	1.000	0.027	0.053	0.996
Docker	1	HMM	50	1000	5	1.000	0.027	0.053	0.996
Docker	2	HMM	50	1000	5	1.000	0.027	0.053	0.996
Docker	30min	HMM	50	1000	10	1.000	0.029	0.055	0.953
Docker	1	HMM	50	1000	10	1.000	0.029	0.055	0.953
Docker	2	HMM	50	1000	10	1.000	0.028	0.055	0.962
Docker	30min	HMM	50	1000	20	0.995	0.041	0.079	0.645
Docker	1	HMM	50	1000	20	0.991	0.041	0.079	0.645
Docker	2	HMM	50	1000	20	0.991	0.040	0.077	0.669
Docker	30min	HMM	50	1000	50	0.979	0.051	0.097	0.508
Docker	1	HMM	50	1000	50	0.979	0.051	0.097	0.509
Docker	2	HMM	50	1000	50	0.979	0.051	0.097	0.509
Docker	30min	HMM	50	1000	100	0.979	0.052	0.098	0.505
Docker	1	HMM	50	1000	100	0.979	0.051	0.098	0.505
Docker	2	HMM	50	1000	100	0.979	0.051	0.098	0.506
Docker	30min	HMM	50	5000	5	1.000	0.035	0.068	0.999
Docker	1	HMM	50	5000	5	1.000	0.035	0.068	0.999
Docker	2	HMM	50	5000	5	1.000	0.035	0.068	0.999
Docker	30min	HMM	50	5000	10	1.000	0.035	0.068	0.999

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	1	HMM	50	5000	10	1.000	0.035	0.068	0.998
Docker	2	HMM	50	5000	10	1.000	0.035	0.068	0.998
Docker	30min	HMM	50	5000	20	1.000	0.035	0.068	0.998
Docker	1	HMM	50	5000	20	1.000	0.035	0.068	0.997
Docker	2	HMM	50	5000	20	1.000	0.035	0.068	0.998
Docker	30min	HMM	50	5000	50	1.000	0.035	0.068	0.992
Docker	1	HMM	50	5000	50	1.000	0.035	0.068	0.991
Docker	2	HMM	50	5000	50	1.000	0.035	0.068	0.993
Docker	30min	HMM	50	5000	100	0.987	0.057	0.107	0.596
Docker	1	HMM	50	5000	100	0.987	0.057	0.107	0.596
Docker	2	HMM	50	5000	100	0.987	0.055	0.104	0.619
Docker	30min	HMM	100	500	5	0.796	0.603	0.686	0.014
Docker	1	HMM	100	500	5	0.796	0.652	0.717	0.011
Docker	2	HMM	100	500	5	0.965	0.670	0.791	0.013
Docker	30min	HMM	100	500	10	0.796	0.783	0.789	0.006
Docker	1	HMM	100	500	10	0.796	0.813	0.804	0.005
Docker	2	HMM	100	500	10	0.965	0.838	0.897	0.005
Docker	30min	HMM	100	500	20	0.796	0.863	0.828	0.003
Docker	1	HMM	100	500	20	0.796	0.865	0.829	0.003
Docker	2	HMM	100	500	20	0.965	0.885	0.923	0.003
Docker	30min	HMM	100	500	50	0.683	0.884	0.771	0.002
Docker	1	HMM	100	500	50	0.767	0.894	0.826	0.002
Docker	2	HMM	100	500	50	0.965	0.913	0.938	0.002
Docker	30min	HMM	100	500	100	0.642	0.892	0.747	0.002
Docker	1	HMM	100	500	100	0.642	0.891	0.747	0.002
Docker	2	HMM	100	500	100	0.965	0.923	0.944	0.002
Docker	30min	HMM	100	1000	5	0.793	0.494	0.608	0.023
Docker	1	HMM	100	1000	5	0.793	0.546	0.647	0.018
Docker	2	HMM	100	1000	5	0.959	0.569	0.714	0.020
Docker	30min	HMM	100	1000	10	0.793	0.600	0.683	0.015
Docker	1	HMM	100	1000	10	0.793	0.642	0.709	0.012
Docker	2	HMM	100	1000	10	0.959	0.654	0.778	0.014
Docker	30min	HMM	100	1000	20	0.793	0.799	0.796	0.006
Docker	1	HMM	100	1000	20	0.793	0.811	0.802	0.005
Docker	2	HMM	100	1000	20	0.959	0.827	0.888	0.006
Docker	30min	HMM	100	1000	50	0.710	0.871	0.783	0.003
Docker	1	HMM	100	1000	50	0.793	0.881	0.835	0.003
Docker	2	HMM	100	1000	50	0.959	0.888	0.922	0.003
Docker	30min	HMM	100	1000	100	0.655	0.884	0.752	0.002
Docker	1	HMM	100	1000	100	0.712	0.892	0.792	0.002
Docker	2	HMM	100	1000	100	0.959	0.910	0.934	0.003
Docker	30min	HMM	100	5000	5	0.813	0.319	0.458	0.063
Docker	1	HMM	100	5000	5	0.813	0.401	0.537	0.044
Docker	2	HMM	100	5000	5	0.973	0.422	0.589	0.048
Docker	30min	HMM	100	5000	10	0.813	0.421	0.555	0.041
Docker	1	HMM	100	5000	10	0.813	0.471	0.597	0.033
Docker	2	HMM	100	5000	10	0.973	0.492	0.653	0.037
Docker	30min	HMM	100	5000	20	0.813	0.449	0.578	0.036
Docker	1	HMM	100	5000	20	0.813	0.490	0.612	0.031
Docker	2	HMM	100	5000	20	0.973	0.509	0.668	0.034
Docker	30min	HMM	100	5000	50	0.800	0.541	0.645	0.025
Docker	1	HMM	100	5000	50	0.800	0.563	0.661	0.022
Docker	2	HMM	100	5000	50	0.973	0.582	0.728	0.025
Docker	30min	HMM	100	5000	100	0.787	0.702	0.742	0.012
Docker	1	HMM	100	5000	100	0.787	0.678	0.728	0.014
Docker	2	HMM	100	5000	100	0.947	0.693	0.800	0.015
Docker	30min	HMM	125	500	5	0.654	0.580	0.615	0.013
Docker	1	HMM	125	500	5	0.683	0.636	0.659	0.010
Docker	2	HMM	125	500	5	0.824	0.675	0.742	0.010
Docker	30min	HMM	125	500	10	0.654	0.755	0.701	0.006
Docker	1	HMM	125	500	10	0.683	0.794	0.735	0.005
Docker	2	HMM	125	500	10	0.824	0.822	0.823	0.005
Docker	30min	HMM	125	500	20	0.654	0.844	0.737	0.003

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	1	HMM	125	500	20	0.654	0.846	0.738	0.003
Docker	2	HMM	125	500	20	0.796	0.868	0.830	0.003
Docker	30min	HMM	125	500	50	0.626	0.882	0.732	0.002
Docker	1	HMM	125	500	50	0.654	0.883	0.752	0.002
Docker	2	HMM	125	500	50	0.683	0.885	0.771	0.002
Docker	30min	HMM	125	500	100	0.614	0.895	0.728	0.002
Docker	1	HMM	125	500	100	0.642	0.898	0.749	0.002
Docker	2	HMM	125	500	100	0.625	0.892	0.735	0.002
Docker	30min	HMM	125	1000	5	0.655	0.470	0.548	0.021
Docker	1	HMM	125	1000	5	0.738	0.547	0.628	0.017
Docker	2	HMM	125	1000	5	0.821	0.549	0.658	0.019
Docker	30min	HMM	125	1000	10	0.655	0.570	0.609	0.014
Docker	1	HMM	125	1000	10	0.684	0.626	0.654	0.011
Docker	2	HMM	125	1000	10	0.821	0.659	0.731	0.012
Docker	30min	HMM	125	1000	20	0.655	0.777	0.711	0.005
Docker	1	HMM	125	1000	20	0.655	0.790	0.716	0.005
Docker	2	HMM	125	1000	20	0.793	0.817	0.805	0.005
Docker	30min	HMM	125	1000	50	0.655	0.866	0.746	0.003
Docker	1	HMM	125	1000	50	0.655	0.864	0.745	0.003
Docker	2	HMM	125	1000	50	0.793	0.883	0.836	0.003
Docker	30min	HMM	125	1000	100	0.628	0.888	0.735	0.002
Docker	1	HMM	125	1000	100	0.655	0.890	0.755	0.002
Docker	2	HMM	125	1000	100	0.655	0.884	0.752	0.002
Docker	30min	HMM	125	5000	5	0.680	0.303	0.419	0.057
Docker	1	HMM	125	5000	5	0.813	0.419	0.553	0.041
Docker	2	HMM	125	5000	5	0.840	0.414	0.555	0.043
Docker	30min	HMM	125	5000	10	0.680	0.392	0.498	0.038
Docker	1	HMM	125	5000	10	0.760	0.465	0.577	0.032
Docker	2	HMM	125	5000	10	0.840	0.468	0.601	0.035
Docker	30min	HMM	125	5000	20	0.680	0.425	0.523	0.033
Docker	1	HMM	125	5000	20	0.707	0.473	0.567	0.029
Docker	2	HMM	125	5000	20	0.840	0.486	0.616	0.032
Docker	30min	HMM	125	5000	50	0.653	0.516	0.576	0.022
Docker	1	HMM	125	5000	50	0.680	0.545	0.605	0.021
Docker	2	HMM	125	5000	50	0.827	0.577	0.679	0.022
Docker	30min	HMM	125	5000	100	0.653	0.681	0.667	0.011
Docker	1	HMM	125	5000	100	0.653	0.676	0.664	0.011
Docker	2	HMM	125	5000	100	0.787	0.702	0.742	0.012
Docker	30min	HMM	150	500	5	0.626	0.576	0.600	0.012
Docker	1	HMM	150	500	5	0.654	0.638	0.646	0.010
Docker	2	HMM	150	500	5	0.683	0.645	0.663	0.010
Docker	30min	HMM	150	500	10	0.626	0.761	0.687	0.005
Docker	1	HMM	150	500	10	0.654	0.802	0.721	0.004
Docker	2	HMM	150	500	10	0.654	0.795	0.718	0.004
Docker	30min	HMM	150	500	20	0.626	0.845	0.719	0.003
Docker	1	HMM	150	500	20	0.654	0.850	0.739	0.003
Docker	2	HMM	150	500	20	0.654	0.844	0.737	0.003
Docker	30min	HMM	150	500	50	0.626	0.889	0.734	0.002
Docker	1	HMM	150	500	50	0.626	0.886	0.734	0.002
Docker	2	HMM	150	500	50	0.654	0.882	0.751	0.002
Docker	30min	HMM	150	500	100	0.599	0.894	0.717	0.002
Docker	1	HMM	150	500	100	0.599	0.893	0.717	0.002
Docker	2	HMM	150	500	100	0.588	0.891	0.708	0.002
Docker	30min	HMM	150	1000	5	0.628	0.476	0.542	0.019
Docker	1	HMM	150	1000	5	0.655	0.534	0.589	0.016
Docker	2	HMM	150	1000	5	0.710	0.547	0.618	0.016
Docker	30min	HMM	150	1000	10	0.628	0.572	0.599	0.013
Docker	1	HMM	150	1000	10	0.655	0.632	0.644	0.011
Docker	2	HMM	150	1000	10	0.684	0.627	0.655	0.011
Docker	30min	HMM	150	1000	20	0.628	0.778	0.695	0.005
Docker	1	HMM	150	1000	20	0.655	0.795	0.718	0.005
Docker	2	HMM	150	1000	20	0.655	0.790	0.716	0.005
Docker	30min	HMM	150	1000	50	0.628	0.875	0.731	0.003

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	1	HMM	150	1000	50	0.628	0.873	0.730	0.003
Docker	2	HMM	150	1000	50	0.655	0.868	0.747	0.003
Docker	30min	HMM	150	1000	100	0.628	0.892	0.737	0.002
Docker	1	HMM	150	1000	100	0.628	0.892	0.737	0.002
Docker	2	HMM	150	1000	100	0.655	0.888	0.754	0.002
Docker	30min	HMM	150	5000	5	0.653	0.301	0.412	0.055
Docker	1	HMM	150	5000	5	0.680	0.382	0.489	0.040
Docker	2	HMM	150	5000	5	0.787	0.405	0.535	0.042
Docker	30min	HMM	150	5000	10	0.653	0.394	0.491	0.037
Docker	1	HMM	150	5000	10	0.680	0.445	0.538	0.031
Docker	2	HMM	150	5000	10	0.707	0.436	0.539	0.033
Docker	30min	HMM	150	5000	20	0.640	0.421	0.508	0.032
Docker	1	HMM	150	5000	20	0.667	0.467	0.549	0.028
Docker	2	HMM	150	5000	20	0.707	0.471	0.565	0.029
Docker	30min	HMM	150	5000	50	0.627	0.553	0.588	0.018
Docker	1	HMM	150	5000	50	0.653	0.587	0.618	0.017
Docker	2	HMM	150	5000	50	0.653	0.560	0.603	0.019
Docker	30min	HMM	150	5000	100	0.627	0.681	0.653	0.011
Docker	1	HMM	150	5000	100	0.653	0.690	0.671	0.011
Docker	2	HMM	150	5000	100	0.653	0.676	0.664	0.011
Docker	6	BoSC	3	500	5	1.000	0.769	0.869	0.007
Docker	12	BoSC	3	500	5	1.000	0.770	0.870	0.007
Docker	24	BoSC	3	500	5	1.000	0.770	0.870	0.007
Docker	6	BoSC	3	500	10	1.000	0.828	0.906	0.005
Docker	12	BoSC	3	500	10	1.000	0.829	0.906	0.005
Docker	24	BoSC	3	500	10	1.000	0.830	0.907	0.005
Docker	6	BoSC	3	500	20	0.990	0.872	0.927	0.004
Docker	12	BoSC	3	500	20	0.987	0.878	0.930	0.003
Docker	24	BoSC	3	500	20	0.982	0.878	0.927	0.003
Docker	6	BoSC	3	500	50	0.975	0.910	0.941	0.002
Docker	12	BoSC	3	500	50	0.975	0.910	0.941	0.002
Docker	24	BoSC	3	500	50	0.975	0.910	0.941	0.002
Docker	6	BoSC	3	500	100	0.975	0.914	0.944	0.002
Docker	12	BoSC	3	500	100	0.975	0.914	0.944	0.002
Docker	24	BoSC	3	500	100	0.975	0.914	0.944	0.002
Docker	6	BoSC	3	1000	5	1.000	0.682	0.811	0.012
Docker	12	BoSC	3	1000	5	1.000	0.684	0.812	0.011
Docker	24	BoSC	3	1000	5	1.000	0.685	0.813	0.011
Docker	6	BoSC	3	1000	10	1.000	0.755	0.860	0.008
Docker	12	BoSC	3	1000	10	1.000	0.755	0.861	0.008
Docker	24	BoSC	3	1000	10	1.000	0.756	0.861	0.008
Docker	6	BoSC	3	1000	20	0.990	0.829	0.902	0.005
Docker	12	BoSC	3	1000	20	0.987	0.839	0.907	0.005
Docker	24	BoSC	3	1000	20	0.981	0.840	0.905	0.005
Docker	6	BoSC	3	1000	50	0.975	0.903	0.938	0.003
Docker	12	BoSC	3	1000	50	0.975	0.903	0.938	0.003
Docker	24	BoSC	3	1000	50	0.975	0.904	0.938	0.003
Docker	6	BoSC	3	1000	100	0.975	0.908	0.940	0.002
Docker	12	BoSC	3	1000	100	0.975	0.908	0.940	0.002
Docker	24	BoSC	3	1000	100	0.975	0.908	0.940	0.002
Docker	6	BoSC	3	5000	5	1.000	0.450	0.621	0.031
Docker	12	BoSC	3	5000	5	1.000	0.455	0.625	0.030
Docker	24	BoSC	3	5000	5	1.000	0.456	0.627	0.030
Docker	6	BoSC	3	5000	10	1.000	0.519	0.684	0.024
Docker	12	BoSC	3	5000	10	1.000	0.520	0.684	0.023
Docker	24	BoSC	3	5000	10	1.000	0.521	0.685	0.023
Docker	6	BoSC	3	5000	20	1.000	0.602	0.752	0.017
Docker	12	BoSC	3	5000	20	1.000	0.611	0.759	0.016
Docker	24	BoSC	3	5000	20	1.000	0.613	0.760	0.016
Docker	6	BoSC	3	5000	50	0.970	0.847	0.904	0.004
Docker	12	BoSC	3	5000	50	0.970	0.848	0.905	0.004
Docker	24	BoSC	3	5000	50	0.970	0.848	0.905	0.004
Docker	6	BoSC	3	5000	100	0.970	0.876	0.920	0.004

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	12	BoSC	3	5000	100	0.970	0.876	0.920	0.004
Docker	24	BoSC	3	5000	100	0.970	0.876	0.920	0.003
Docker	6	BoSC	4	500	5	1.000	0.746	0.855	0.008
Docker	12	BoSC	4	500	5	1.000	0.749	0.857	0.008
Docker	24	BoSC	4	500	5	1.000	0.751	0.858	0.008
Docker	6	BoSC	4	500	10	1.000	0.809	0.895	0.006
Docker	12	BoSC	4	500	10	1.000	0.810	0.895	0.006
Docker	24	BoSC	4	500	10	1.000	0.811	0.896	0.006
Docker	6	BoSC	4	500	20	0.999	0.846	0.916	0.004
Docker	12	BoSC	4	500	20	0.999	0.847	0.917	0.004
Docker	24	BoSC	4	500	20	0.995	0.847	0.915	0.004
Docker	6	BoSC	4	500	50	0.975	0.905	0.939	0.003
Docker	12	BoSC	4	500	50	0.975	0.906	0.939	0.003
Docker	24	BoSC	4	500	50	0.975	0.906	0.939	0.003
Docker	6	BoSC	4	500	100	0.975	0.913	0.943	0.002
Docker	12	BoSC	4	500	100	0.975	0.913	0.943	0.002
Docker	24	BoSC	4	500	100	0.975	0.913	0.943	0.002
Docker	6	BoSC	4	1000	5	1.000	0.657	0.793	0.013
Docker	12	BoSC	4	1000	5	1.000	0.662	0.797	0.013
Docker	24	BoSC	4	1000	5	1.000	0.664	0.798	0.013
Docker	6	BoSC	4	1000	10	1.000	0.733	0.846	0.009
Docker	12	BoSC	4	1000	10	1.000	0.735	0.847	0.009
Docker	24	BoSC	4	1000	10	1.000	0.736	0.848	0.009
Docker	6	BoSC	4	1000	20	0.999	0.788	0.881	0.007
Docker	12	BoSC	4	1000	20	0.999	0.789	0.882	0.007
Docker	24	BoSC	4	1000	20	0.996	0.789	0.881	0.007
Docker	6	BoSC	4	1000	50	0.975	0.888	0.929	0.003
Docker	12	BoSC	4	1000	50	0.975	0.889	0.930	0.003
Docker	24	BoSC	4	1000	50	0.975	0.889	0.930	0.003
Docker	6	BoSC	4	1000	100	0.975	0.906	0.939	0.003
Docker	12	BoSC	4	1000	100	0.975	0.906	0.939	0.002
Docker	24	BoSC	4	1000	100	0.975	0.907	0.939	0.002
Docker	6	BoSC	4	5000	5	1.000	0.416	0.588	0.036
Docker	12	BoSC	4	5000	5	1.000	0.428	0.600	0.034
Docker	24	BoSC	4	5000	5	1.000	0.435	0.607	0.033
Docker	6	BoSC	4	5000	10	1.000	0.494	0.661	0.026
Docker	12	BoSC	4	5000	10	1.000	0.499	0.666	0.026
Docker	24	BoSC	4	5000	10	1.000	0.501	0.668	0.025
Docker	6	BoSC	4	5000	20	1.000	0.548	0.708	0.021
Docker	12	BoSC	4	5000	20	1.000	0.550	0.710	0.021
Docker	24	BoSC	4	5000	20	1.000	0.551	0.710	0.021
Docker	6	BoSC	4	5000	50	0.976	0.778	0.866	0.007
Docker	12	BoSC	4	5000	50	0.973	0.781	0.866	0.007
Docker	24	BoSC	4	5000	50	0.973	0.783	0.867	0.007
Docker	6	BoSC	4	5000	100	0.970	0.855	0.909	0.004
Docker	12	BoSC	4	5000	100	0.970	0.855	0.909	0.004
Docker	24	BoSC	4	5000	100	0.970	0.856	0.909	0.004
Docker	6	BoSC	5	500	5	1.000	0.723	0.839	0.009
Docker	12	BoSC	5	500	5	1.000	0.730	0.844	0.009
Docker	24	BoSC	5	500	5	1.000	0.733	0.846	0.009
Docker	6	BoSC	5	500	10	1.000	0.791	0.884	0.007
Docker	12	BoSC	5	500	10	1.000	0.794	0.885	0.006
Docker	24	BoSC	5	500	10	1.000	0.795	0.886	0.006
Docker	6	BoSC	5	500	20	1.000	0.830	0.907	0.005
Docker	12	BoSC	5	500	20	1.000	0.831	0.908	0.005
Docker	24	BoSC	5	500	20	1.000	0.832	0.908	0.005
Docker	6	BoSC	5	500	50	0.975	0.894	0.933	0.003
Docker	12	BoSC	5	500	50	0.975	0.894	0.933	0.003
Docker	24	BoSC	5	500	50	0.975	0.895	0.933	0.003
Docker	6	BoSC	5	500	100	0.975	0.912	0.942	0.002
Docker	12	BoSC	5	500	100	0.975	0.912	0.943	0.002
Docker	24	BoSC	5	500	100	0.975	0.912	0.943	0.002
Docker	6	BoSC	5	1000	5	1.000	0.626	0.770	0.015

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	12	BoSC	5	1000	5	1.000	0.637	0.778	0.014
Docker	24	BoSC	5	1000	5	1.000	0.642	0.782	0.014
Docker	6	BoSC	5	1000	10	1.000	0.714	0.833	0.010
Docker	12	BoSC	5	1000	10	1.000	0.718	0.836	0.010
Docker	24	BoSC	5	1000	10	1.000	0.721	0.838	0.010
Docker	6	BoSC	5	1000	20	1.000	0.769	0.869	0.007
Docker	12	BoSC	5	1000	20	1.000	0.770	0.870	0.007
Docker	24	BoSC	5	1000	20	1.000	0.771	0.871	0.007
Docker	6	BoSC	5	1000	50	0.975	0.865	0.917	0.004
Docker	12	BoSC	5	1000	50	0.975	0.866	0.917	0.004
Docker	24	BoSC	5	1000	50	0.975	0.866	0.917	0.004
Docker	6	BoSC	5	1000	100	0.975	0.899	0.935	0.003
Docker	12	BoSC	5	1000	100	0.975	0.900	0.936	0.003
Docker	24	BoSC	5	1000	100	0.975	0.900	0.936	0.003
Docker	6	BoSC	5	5000	5	1.000	0.360	0.529	0.045
Docker	12	BoSC	5	5000	5	1.000	0.383	0.553	0.041
Docker	24	BoSC	5	5000	5	1.000	0.396	0.567	0.039
Docker	6	BoSC	5	5000	10	1.000	0.460	0.630	0.030
Docker	12	BoSC	5	5000	10	1.000	0.475	0.644	0.028
Docker	24	BoSC	5	5000	10	1.000	0.481	0.650	0.027
Docker	6	BoSC	5	5000	20	1.000	0.529	0.692	0.023
Docker	12	BoSC	5	5000	20	1.000	0.533	0.695	0.022
Docker	24	BoSC	5	5000	20	1.000	0.534	0.696	0.022
Docker	6	BoSC	5	5000	50	0.994	0.662	0.795	0.013
Docker	12	BoSC	5	5000	50	0.991	0.663	0.795	0.013
Docker	24	BoSC	5	5000	50	0.988	0.665	0.795	0.013
Docker	6	BoSC	5	5000	100	0.970	0.825	0.892	0.005
Docker	12	BoSC	5	5000	100	0.970	0.825	0.892	0.005
Docker	24	BoSC	5	5000	100	0.970	0.826	0.892	0.005
Docker	6	BoSC	6	500	5	1.000	0.677	0.807	0.012
Docker	12	BoSC	6	500	5	1.000	0.690	0.816	0.011
Docker	24	BoSC	6	500	5	1.000	0.701	0.824	0.011
Docker	6	BoSC	6	500	10	1.000	0.770	0.870	0.007
Docker	12	BoSC	6	500	10	1.000	0.775	0.874	0.007
Docker	24	BoSC	6	500	10	1.000	0.779	0.876	0.007
Docker	6	BoSC	6	500	20	1.000	0.818	0.900	0.006
Docker	12	BoSC	6	500	20	1.000	0.819	0.901	0.005
Docker	24	BoSC	6	500	20	1.000	0.820	0.901	0.005
Docker	6	BoSC	6	500	50	0.985	0.882	0.930	0.003
Docker	12	BoSC	6	500	50	0.982	0.882	0.929	0.003
Docker	24	BoSC	6	500	50	0.979	0.882	0.928	0.003
Docker	6	BoSC	6	500	100	0.975	0.906	0.939	0.002
Docker	12	BoSC	6	500	100	0.975	0.907	0.940	0.002
Docker	24	BoSC	6	500	100	0.975	0.907	0.940	0.002
Docker	6	BoSC	6	1000	5	1.000	0.562	0.720	0.019
Docker	12	BoSC	6	1000	5	1.000	0.581	0.735	0.018
Docker	24	BoSC	6	1000	5	1.000	0.599	0.749	0.017
Docker	6	BoSC	6	1000	10	1.000	0.679	0.809	0.012
Docker	12	BoSC	6	1000	10	1.000	0.688	0.815	0.011
Docker	24	BoSC	6	1000	10	1.000	0.696	0.821	0.011
Docker	6	BoSC	6	1000	20	1.000	0.753	0.859	0.008
Docker	12	BoSC	6	1000	20	1.000	0.756	0.861	0.008
Docker	24	BoSC	6	1000	20	1.000	0.758	0.862	0.008
Docker	6	BoSC	6	1000	50	0.984	0.851	0.913	0.004
Docker	12	BoSC	6	1000	50	0.981	0.851	0.912	0.004
Docker	24	BoSC	6	1000	50	0.979	0.851	0.911	0.004
Docker	6	BoSC	6	1000	100	0.975	0.883	0.926	0.003
Docker	12	BoSC	6	1000	100	0.975	0.884	0.927	0.003
Docker	24	BoSC	6	1000	100	0.975	0.884	0.927	0.003
Docker	6	BoSC	6	5000	5	1.000	0.291	0.451	0.062
Docker	12	BoSC	6	5000	5	1.000	0.315	0.480	0.055
Docker	24	BoSC	6	5000	5	1.000	0.338	0.505	0.050
Docker	6	BoSC	6	5000	10	1.000	0.400	0.572	0.038

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	12	BoSC	6	5000	10	1.000	0.420	0.591	0.035
Docker	24	BoSC	6	5000	10	1.000	0.438	0.609	0.033
Docker	6	BoSC	6	5000	20	1.000	0.499	0.666	0.026
Docker	12	BoSC	6	5000	20	1.000	0.512	0.677	0.024
Docker	24	BoSC	6	5000	20	1.000	0.518	0.683	0.024
Docker	6	BoSC	6	5000	50	1.000	0.640	0.781	0.014
Docker	12	BoSC	6	5000	50	1.000	0.643	0.782	0.014
Docker	24	BoSC	6	5000	50	0.999	0.646	0.784	0.014
Docker	6	BoSC	6	5000	100	0.970	0.773	0.861	0.007
Docker	12	BoSC	6	5000	100	0.970	0.775	0.862	0.007
Docker	24	BoSC	6	5000	100	0.970	0.776	0.862	0.007
Docker	6	STIDE	3	500	5	1.000	0.761	0.864	0.008
Docker	12	STIDE	3	500	5	1.000	0.762	0.865	0.008
Docker	24	STIDE	3	500	5	1.000	0.763	0.866	0.008
Docker	6	STIDE	3	500	10	1.000	0.822	0.903	0.005
Docker	12	STIDE	3	500	10	1.000	0.823	0.903	0.005
Docker	24	STIDE	3	500	10	1.000	0.823	0.903	0.005
Docker	6	STIDE	3	500	20	0.994	0.859	0.922	0.004
Docker	12	STIDE	3	500	20	0.991	0.861	0.921	0.004
Docker	24	STIDE	3	500	20	0.987	0.869	0.924	0.004
Docker	6	STIDE	3	500	50	0.975	0.909	0.941	0.002
Docker	12	STIDE	3	500	50	0.975	0.909	0.941	0.002
Docker	24	STIDE	3	500	50	0.975	0.909	0.941	0.002
Docker	6	STIDE	3	500	100	0.975	0.913	0.943	0.002
Docker	12	STIDE	3	500	100	0.975	0.914	0.943	0.002
Docker	24	STIDE	3	500	100	0.975	0.914	0.944	0.002
Docker	6	STIDE	3	1000	5	1.000	0.674	0.806	0.012
Docker	12	STIDE	3	1000	5	1.000	0.676	0.807	0.012
Docker	24	STIDE	3	1000	5	1.000	0.677	0.808	0.012
Docker	6	STIDE	3	1000	10	1.000	0.748	0.856	0.008
Docker	12	STIDE	3	1000	10	1.000	0.749	0.856	0.008
Docker	24	STIDE	3	1000	10	1.000	0.749	0.857	0.008
Docker	6	STIDE	3	1000	20	0.994	0.805	0.889	0.006
Docker	12	STIDE	3	1000	20	0.991	0.807	0.890	0.006
Docker	24	STIDE	3	1000	20	0.987	0.822	0.897	0.005
Docker	6	STIDE	3	1000	50	0.975	0.902	0.937	0.003
Docker	12	STIDE	3	1000	50	0.975	0.902	0.937	0.003
Docker	24	STIDE	3	1000	50	0.975	0.902	0.937	0.003
Docker	6	STIDE	3	1000	100	0.975	0.908	0.940	0.002
Docker	12	STIDE	3	1000	100	0.975	0.908	0.940	0.002
Docker	24	STIDE	3	1000	100	0.975	0.908	0.940	0.002
Docker	6	STIDE	3	5000	5	1.000	0.443	0.614	0.032
Docker	12	STIDE	3	5000	5	1.000	0.446	0.617	0.032
Docker	24	STIDE	3	5000	5	1.000	0.450	0.621	0.031
Docker	6	STIDE	3	5000	10	1.000	0.511	0.676	0.024
Docker	12	STIDE	3	5000	10	1.000	0.512	0.677	0.024
Docker	24	STIDE	3	5000	10	1.000	0.513	0.678	0.024
Docker	6	STIDE	3	5000	20	1.000	0.574	0.730	0.019
Docker	12	STIDE	3	5000	20	1.000	0.578	0.732	0.019
Docker	24	STIDE	3	5000	20	1.000	0.592	0.744	0.018
Docker	6	STIDE	3	5000	50	0.970	0.838	0.899	0.005
Docker	12	STIDE	3	5000	50	0.970	0.839	0.900	0.005
Docker	24	STIDE	3	5000	50	0.970	0.839	0.900	0.005
Docker	6	STIDE	3	5000	100	0.970	0.868	0.916	0.004
Docker	12	STIDE	3	5000	100	0.970	0.869	0.917	0.004
Docker	24	STIDE	3	5000	100	0.970	0.870	0.917	0.004
Docker	6	STIDE	4	500	5	1.000	0.727	0.842	0.009
Docker	12	STIDE	4	500	5	1.000	0.732	0.845	0.009
Docker	24	STIDE	4	500	5	1.000	0.735	0.848	0.009
Docker	6	STIDE	4	500	10	1.000	0.793	0.885	0.006
Docker	12	STIDE	4	500	10	1.000	0.795	0.886	0.006
Docker	24	STIDE	4	500	10	1.000	0.797	0.887	0.006
Docker	6	STIDE	4	500	20	0.999	0.833	0.908	0.005

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	12	STIDE	4	500	20	0.999	0.833	0.909	0.005
Docker	24	STIDE	4	500	20	0.999	0.835	0.909	0.005
Docker	6	STIDE	4	500	50	0.975	0.896	0.934	0.003
Docker	12	STIDE	4	500	50	0.975	0.897	0.934	0.003
Docker	24	STIDE	4	500	50	0.975	0.898	0.935	0.003
Docker	6	STIDE	4	500	100	0.975	0.912	0.943	0.002
Docker	12	STIDE	4	500	100	0.975	0.912	0.943	0.002
Docker	24	STIDE	4	500	100	0.975	0.912	0.943	0.002
Docker	6	STIDE	4	1000	5	1.000	0.635	0.777	0.014
Docker	12	STIDE	4	1000	5	1.000	0.642	0.782	0.014
Docker	24	STIDE	4	1000	5	1.000	0.647	0.786	0.014
Docker	6	STIDE	4	1000	10	1.000	0.716	0.834	0.010
Docker	12	STIDE	4	1000	10	1.000	0.720	0.837	0.010
Docker	24	STIDE	4	1000	10	1.000	0.722	0.839	0.010
Docker	6	STIDE	4	1000	20	0.999	0.771	0.870	0.007
Docker	12	STIDE	4	1000	20	0.999	0.773	0.871	0.007
Docker	24	STIDE	4	1000	20	0.999	0.774	0.872	0.007
Docker	6	STIDE	4	1000	50	0.975	0.868	0.918	0.004
Docker	12	STIDE	4	1000	50	0.975	0.869	0.919	0.004
Docker	24	STIDE	4	1000	50	0.975	0.870	0.919	0.004
Docker	6	STIDE	4	1000	100	0.975	0.902	0.937	0.003
Docker	12	STIDE	4	1000	100	0.975	0.902	0.937	0.003
Docker	24	STIDE	4	1000	100	0.975	0.903	0.937	0.003
Docker	6	STIDE	4	5000	5	1.000	0.384	0.555	0.041
Docker	12	STIDE	4	5000	5	1.000	0.398	0.569	0.038
Docker	24	STIDE	4	5000	5	1.000	0.409	0.580	0.037
Docker	6	STIDE	4	5000	10	1.000	0.473	0.642	0.028
Docker	12	STIDE	4	5000	10	1.000	0.482	0.651	0.027
Docker	24	STIDE	4	5000	10	1.000	0.487	0.655	0.027
Docker	6	STIDE	4	5000	20	1.000	0.533	0.695	0.022
Docker	12	STIDE	4	5000	20	1.000	0.536	0.698	0.022
Docker	24	STIDE	4	5000	20	1.000	0.537	0.699	0.022
Docker	6	STIDE	4	5000	50	0.979	0.691	0.810	0.011
Docker	12	STIDE	4	5000	50	0.975	0.703	0.817	0.010
Docker	24	STIDE	4	5000	50	0.975	0.737	0.840	0.009
Docker	6	STIDE	4	5000	100	0.970	0.829	0.894	0.005
Docker	12	STIDE	4	5000	100	0.970	0.830	0.894	0.005
Docker	24	STIDE	4	5000	100	0.970	0.831	0.895	0.005
Docker	6	STIDE	5	500	5	1.000	0.655	0.792	0.013
Docker	12	STIDE	5	500	5	1.000	0.670	0.803	0.012
Docker	24	STIDE	5	500	5	1.000	0.684	0.812	0.011
Docker	6	STIDE	5	500	10	1.000	0.755	0.860	0.008
Docker	12	STIDE	5	500	10	1.000	0.761	0.864	0.008
Docker	24	STIDE	5	500	10	1.000	0.765	0.867	0.008
Docker	6	STIDE	5	500	20	1.000	0.805	0.892	0.006
Docker	12	STIDE	5	500	20	1.000	0.807	0.893	0.006
Docker	24	STIDE	5	500	20	1.000	0.809	0.894	0.006
Docker	6	STIDE	5	500	50	0.977	0.874	0.923	0.003
Docker	12	STIDE	5	500	50	0.977	0.874	0.923	0.003
Docker	24	STIDE	5	500	50	0.977	0.875	0.923	0.003
Docker	6	STIDE	5	500	100	0.975	0.903	0.938	0.003
Docker	12	STIDE	5	500	100	0.975	0.903	0.938	0.003
Docker	24	STIDE	5	500	100	0.975	0.904	0.938	0.003
Docker	6	STIDE	5	1000	5	1.000	0.541	0.702	0.021
Docker	12	STIDE	5	1000	5	1.000	0.564	0.721	0.019
Docker	24	STIDE	5	1000	5	1.000	0.583	0.737	0.018
Docker	6	STIDE	5	1000	10	1.000	0.665	0.799	0.013
Docker	12	STIDE	5	1000	10	1.000	0.672	0.804	0.012
Docker	24	STIDE	5	1000	10	1.000	0.681	0.810	0.012
Docker	6	STIDE	5	1000	20	1.000	0.735	0.847	0.009
Docker	12	STIDE	5	1000	20	1.000	0.739	0.850	0.009
Docker	24	STIDE	5	1000	20	1.000	0.742	0.852	0.009
Docker	6	STIDE	5	1000	50	0.977	0.838	0.902	0.005

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	12	STIDE	5	1000	50	0.977	0.840	0.903	0.005
Docker	24	STIDE	5	1000	50	0.977	0.841	0.904	0.005
Docker	6	STIDE	5	1000	100	0.975	0.872	0.921	0.004
Docker	12	STIDE	5	1000	100	0.975	0.873	0.921	0.004
Docker	24	STIDE	5	1000	100	0.975	0.874	0.922	0.003
Docker	6	STIDE	5	5000	5	1.000	0.255	0.406	0.074
Docker	12	STIDE	5	5000	5	1.000	0.297	0.458	0.060
Docker	24	STIDE	5	5000	5	1.000	0.327	0.493	0.052
Docker	6	STIDE	5	5000	10	1.000	0.395	0.566	0.039
Docker	12	STIDE	5	5000	10	1.000	0.417	0.589	0.036
Docker	24	STIDE	5	5000	10	1.000	0.436	0.608	0.033
Docker	6	STIDE	5	5000	20	1.000	0.486	0.654	0.027
Docker	12	STIDE	5	5000	20	1.000	0.496	0.663	0.026
Docker	24	STIDE	5	5000	20	1.000	0.508	0.673	0.025
Docker	6	STIDE	5	5000	50	0.998	0.627	0.770	0.015
Docker	12	STIDE	5	5000	50	0.996	0.632	0.774	0.015
Docker	24	STIDE	5	5000	50	0.994	0.635	0.775	0.015
Docker	6	STIDE	5	5000	100	0.970	0.742	0.841	0.009
Docker	12	STIDE	5	5000	100	0.970	0.745	0.842	0.008
Docker	24	STIDE	5	5000	100	0.970	0.747	0.844	0.008
Docker	6	STIDE	6	500	5	1.000	0.512	0.677	0.024
Docker	12	STIDE	6	500	5	1.000	0.567	0.724	0.019
Docker	24	STIDE	6	500	5	1.000	0.599	0.749	0.017
Docker	6	STIDE	6	500	10	1.000	0.693	0.819	0.011
Docker	12	STIDE	6	500	10	1.000	0.707	0.829	0.010
Docker	24	STIDE	6	500	10	1.000	0.721	0.838	0.010
Docker	6	STIDE	6	500	20	1.000	0.771	0.871	0.007
Docker	12	STIDE	6	500	20	1.000	0.775	0.873	0.007
Docker	24	STIDE	6	500	20	1.000	0.778	0.875	0.007
Docker	6	STIDE	6	500	50	0.985	0.849	0.912	0.004
Docker	12	STIDE	6	500	50	0.985	0.851	0.913	0.004
Docker	24	STIDE	6	500	50	0.985	0.852	0.914	0.004
Docker	6	STIDE	6	500	100	0.975	0.884	0.927	0.003
Docker	12	STIDE	6	500	100	0.975	0.885	0.928	0.003
Docker	24	STIDE	6	500	100	0.975	0.886	0.928	0.003
Docker	6	STIDE	6	1000	5	1.000	0.366	0.536	0.043
Docker	12	STIDE	6	1000	5	1.000	0.438	0.609	0.032
Docker	24	STIDE	6	1000	5	1.000	0.479	0.647	0.027
Docker	6	STIDE	6	1000	10	1.000	0.579	0.733	0.018
Docker	12	STIDE	6	1000	10	1.000	0.606	0.755	0.016
Docker	24	STIDE	6	1000	10	1.000	0.629	0.772	0.015
Docker	6	STIDE	6	1000	20	1.000	0.694	0.819	0.011
Docker	12	STIDE	6	1000	20	1.000	0.700	0.823	0.011
Docker	24	STIDE	6	1000	20	1.000	0.708	0.829	0.010
Docker	6	STIDE	6	1000	50	0.984	0.804	0.885	0.006
Docker	12	STIDE	6	1000	50	0.984	0.807	0.887	0.006
Docker	24	STIDE	6	1000	50	0.984	0.809	0.888	0.006
Docker	6	STIDE	6	1000	100	0.975	0.843	0.904	0.005
Docker	12	STIDE	6	1000	100	0.975	0.844	0.904	0.004
Docker	24	STIDE	6	1000	100	0.975	0.845	0.905	0.004
Docker	6	STIDE	6	5000	5	1.000	0.108	0.194	0.211
Docker	12	STIDE	6	5000	5	1.000	0.165	0.283	0.129
Docker	24	STIDE	6	5000	5	1.000	0.209	0.346	0.096
Docker	6	STIDE	6	5000	10	1.000	0.239	0.385	0.081
Docker	12	STIDE	6	5000	10	1.000	0.309	0.472	0.057
Docker	24	STIDE	6	5000	10	1.000	0.347	0.515	0.048
Docker	6	STIDE	6	5000	20	1.000	0.413	0.585	0.036
Docker	12	STIDE	6	5000	20	1.000	0.441	0.612	0.032
Docker	24	STIDE	6	5000	20	1.000	0.464	0.634	0.029
Docker	6	STIDE	6	5000	50	1.000	0.583	0.737	0.018
Docker	12	STIDE	6	5000	50	1.000	0.596	0.747	0.017
Docker	24	STIDE	6	5000	50	1.000	0.605	0.754	0.017
Docker	6	STIDE	6	5000	100	0.970	0.702	0.814	0.010

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
Docker	12	STIDE	6	5000	100	0.970	0.706	0.817	0.010
Docker	24	STIDE	6	5000	100	0.970	0.710	0.820	0.010
OS	6	BoSC	3	500	5	0.727	0.843	0.781	0.001
OS	12	BoSC	3	500	5	0.666	0.910	0.769	0.001
OS	24	BoSC	3	500	5	0.666	0.925	0.774	0.000
OS	6	BoSC	3	500	10	0.666	0.934	0.777	0.000
OS	12	BoSC	3	500	10	0.666	0.954	0.784	0.000
OS	24	BoSC	3	500	10	0.666	0.957	0.785	0.000
OS	6	BoSC	3	500	20	0.635	0.973	0.769	0.000
OS	12	BoSC	3	500	20	0.635	0.976	0.770	0.000
OS	24	BoSC	3	500	20	0.629	0.977	0.766	0.000
OS	6	BoSC	3	500	50	0.593	0.983	0.740	0.000
OS	12	BoSC	3	500	50	0.593	0.984	0.740	0.000
OS	24	BoSC	3	500	50	0.593	0.984	0.740	0.000
OS	6	BoSC	3	500	100	0.593	0.987	0.741	0.000
OS	12	BoSC	3	500	100	0.593	0.987	0.741	0.000
OS	24	BoSC	3	500	100	0.593	0.987	0.741	0.000
OS	6	BoSC	3	1000	5	0.728	0.767	0.747	0.002
OS	12	BoSC	3	1000	5	0.666	0.855	0.749	0.001
OS	24	BoSC	3	1000	5	0.666	0.884	0.760	0.001
OS	6	BoSC	3	1000	10	0.666	0.870	0.755	0.001
OS	12	BoSC	3	1000	10	0.666	0.918	0.772	0.001
OS	24	BoSC	3	1000	10	0.666	0.927	0.775	0.000
OS	6	BoSC	3	1000	20	0.659	0.948	0.778	0.000
OS	12	BoSC	3	1000	20	0.659	0.961	0.782	0.000
OS	24	BoSC	3	1000	20	0.653	0.962	0.778	0.000
OS	6	BoSC	3	1000	50	0.593	0.978	0.738	0.000
OS	12	BoSC	3	1000	50	0.593	0.979	0.739	0.000
OS	24	BoSC	3	1000	50	0.593	0.979	0.739	0.000
OS	6	BoSC	3	1000	100	0.593	0.983	0.740	0.000
OS	12	BoSC	3	1000	100	0.593	0.983	0.740	0.000
OS	24	BoSC	3	1000	100	0.593	0.984	0.740	0.000
OS	6	BoSC	3	5000	5	0.731	0.518	0.606	0.006
OS	12	BoSC	3	5000	5	0.670	0.616	0.642	0.004
OS	24	BoSC	3	5000	5	0.670	0.678	0.674	0.003
OS	6	BoSC	3	5000	10	0.670	0.635	0.652	0.004
OS	12	BoSC	3	5000	10	0.670	0.729	0.698	0.002
OS	24	BoSC	3	5000	10	0.670	0.770	0.716	0.002
OS	6	BoSC	3	5000	20	0.670	0.782	0.721	0.002
OS	12	BoSC	3	5000	20	0.670	0.857	0.752	0.001
OS	24	BoSC	3	5000	20	0.664	0.876	0.755	0.001
OS	6	BoSC	3	5000	50	0.594	0.910	0.719	0.001
OS	12	BoSC	3	5000	50	0.594	0.937	0.727	0.000
OS	24	BoSC	3	5000	50	0.594	0.939	0.728	0.000
OS	6	BoSC	3	5000	100	0.586	0.957	0.727	0.000
OS	12	BoSC	3	5000	100	0.586	0.958	0.728	0.000
OS	24	BoSC	3	5000	100	0.586	0.962	0.728	0.000
OS	6	BoSC	4	500	5	0.748	0.809	0.777	0.002
OS	12	BoSC	4	500	5	0.748	0.862	0.801	0.001
OS	24	BoSC	4	500	5	0.686	0.889	0.775	0.001
OS	6	BoSC	4	500	10	0.666	0.894	0.763	0.001
OS	12	BoSC	4	500	10	0.666	0.913	0.770	0.001
OS	24	BoSC	4	500	10	0.666	0.927	0.775	0.000
OS	6	BoSC	4	500	20	0.666	0.948	0.782	0.000
OS	12	BoSC	4	500	20	0.659	0.953	0.779	0.000
OS	24	BoSC	4	500	20	0.659	0.955	0.780	0.000
OS	6	BoSC	4	500	50	0.593	0.980	0.739	0.000
OS	12	BoSC	4	500	50	0.593	0.981	0.739	0.000
OS	24	BoSC	4	500	50	0.593	0.981	0.739	0.000
OS	6	BoSC	4	500	100	0.593	0.985	0.740	0.000
OS	12	BoSC	4	500	100	0.593	0.985	0.740	0.000
OS	24	BoSC	4	500	100	0.593	0.986	0.740	0.000
OS	6	BoSC	4	1000	5	0.748	0.721	0.735	0.003

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
OS	12	BoSC	4	1000	5	0.748	0.784	0.766	0.002
OS	24	BoSC	4	1000	5	0.687	0.829	0.751	0.001
OS	6	BoSC	4	1000	10	0.666	0.820	0.735	0.001
OS	12	BoSC	4	1000	10	0.666	0.858	0.750	0.001
OS	24	BoSC	4	1000	10	0.666	0.888	0.761	0.001
OS	6	BoSC	4	1000	20	0.666	0.898	0.765	0.001
OS	12	BoSC	4	1000	20	0.666	0.918	0.772	0.001
OS	24	BoSC	4	1000	20	0.666	0.925	0.775	0.001
OS	6	BoSC	4	1000	50	0.593	0.963	0.734	0.000
OS	12	BoSC	4	1000	50	0.593	0.965	0.734	0.000
OS	24	BoSC	4	1000	50	0.593	0.967	0.735	0.000
OS	6	BoSC	4	1000	100	0.593	0.980	0.739	0.000
OS	12	BoSC	4	1000	100	0.593	0.981	0.739	0.000
OS	24	BoSC	4	1000	100	0.593	0.982	0.739	0.000
OS	6	BoSC	4	5000	5	0.752	0.460	0.571	0.008
OS	12	BoSC	4	5000	5	0.752	0.522	0.616	0.007
OS	24	BoSC	4	5000	5	0.690	0.585	0.633	0.005
OS	6	BoSC	4	5000	10	0.670	0.558	0.609	0.005
OS	12	BoSC	4	5000	10	0.670	0.611	0.639	0.004
OS	24	BoSC	4	5000	10	0.670	0.686	0.678	0.003
OS	6	BoSC	4	5000	20	0.670	0.662	0.666	0.003
OS	12	BoSC	4	5000	20	0.670	0.721	0.694	0.002
OS	24	BoSC	4	5000	20	0.670	0.768	0.715	0.002
OS	6	BoSC	4	5000	50	0.609	0.844	0.707	0.001
OS	12	BoSC	4	5000	50	0.609	0.883	0.721	0.001
OS	24	BoSC	4	5000	50	0.609	0.899	0.726	0.001
OS	6	BoSC	4	5000	100	0.594	0.923	0.723	0.000
OS	12	BoSC	4	5000	100	0.586	0.930	0.719	0.000
OS	24	BoSC	4	5000	100	0.586	0.931	0.719	0.000
OS	6	BoSC	5	500	5	0.776	0.766	0.771	0.002
OS	12	BoSC	5	500	5	0.748	0.779	0.763	0.002
OS	24	BoSC	5	500	5	0.718	0.837	0.773	0.001
OS	6	BoSC	5	500	10	0.686	0.868	0.767	0.001
OS	12	BoSC	5	500	10	0.666	0.874	0.756	0.001
OS	24	BoSC	5	500	10	0.666	0.899	0.765	0.001
OS	6	BoSC	5	500	20	0.666	0.924	0.774	0.001
OS	12	BoSC	5	500	20	0.666	0.931	0.776	0.000
OS	24	BoSC	5	500	20	0.666	0.935	0.778	0.000
OS	6	BoSC	5	500	50	0.616	0.969	0.753	0.000
OS	12	BoSC	5	500	50	0.610	0.971	0.749	0.000
OS	24	BoSC	5	500	50	0.598	0.971	0.740	0.000
OS	6	BoSC	5	500	100	0.593	0.984	0.740	0.000
OS	12	BoSC	5	500	100	0.593	0.984	0.740	0.000
OS	24	BoSC	5	500	100	0.593	0.985	0.740	0.000
OS	6	BoSC	5	1000	5	0.776	0.670	0.719	0.004
OS	12	BoSC	5	1000	5	0.748	0.686	0.716	0.003
OS	24	BoSC	5	1000	5	0.719	0.756	0.737	0.002
OS	6	BoSC	5	1000	10	0.687	0.782	0.732	0.002
OS	12	BoSC	5	1000	10	0.666	0.792	0.724	0.002
OS	24	BoSC	5	1000	10	0.666	0.841	0.743	0.001
OS	6	BoSC	5	1000	20	0.666	0.862	0.752	0.001
OS	12	BoSC	5	1000	20	0.666	0.881	0.759	0.001
OS	24	BoSC	5	1000	20	0.666	0.897	0.765	0.001
OS	6	BoSC	5	1000	50	0.654	0.946	0.773	0.000
OS	12	BoSC	5	1000	50	0.648	0.949	0.770	0.000
OS	24	BoSC	5	1000	50	0.642	0.950	0.766	0.000
OS	6	BoSC	5	1000	100	0.593	0.972	0.736	0.000
OS	12	BoSC	5	1000	100	0.593	0.974	0.737	0.000
OS	24	BoSC	5	1000	100	0.593	0.975	0.737	0.000
OS	6	BoSC	5	5000	5	0.779	0.384	0.514	0.012
OS	12	BoSC	5	5000	5	0.752	0.414	0.534	0.010
OS	24	BoSC	5	5000	5	0.722	0.483	0.579	0.007
OS	6	BoSC	5	5000	10	0.690	0.519	0.592	0.006

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
OS	12	BoSC	5	5000	10	0.670	0.533	0.593	0.006
OS	24	BoSC	5	5000	10	0.670	0.614	0.640	0.004
OS	6	BoSC	5	5000	20	0.670	0.609	0.638	0.004
OS	12	BoSC	5	5000	20	0.670	0.640	0.654	0.004
OS	24	BoSC	5	5000	20	0.670	0.703	0.686	0.003
OS	6	BoSC	5	5000	50	0.664	0.779	0.717	0.002
OS	12	BoSC	5	5000	50	0.654	0.804	0.721	0.002
OS	24	BoSC	5	5000	50	0.648	0.832	0.729	0.001
OS	6	BoSC	5	5000	100	0.594	0.886	0.711	0.001
OS	12	BoSC	5	5000	100	0.594	0.897	0.715	0.001
OS	24	BoSC	5	5000	100	0.594	0.901	0.716	0.001
OS	6	BoSC	6	500	5	0.890	0.727	0.800	0.003
OS	12	BoSC	6	500	5	0.830	0.740	0.782	0.003
OS	24	BoSC	6	500	5	0.776	0.781	0.778	0.002
OS	6	BoSC	6	500	10	0.686	0.822	0.748	0.001
OS	12	BoSC	6	500	10	0.686	0.837	0.754	0.001
OS	24	BoSC	6	500	10	0.686	0.862	0.764	0.001
OS	6	BoSC	6	500	20	0.666	0.900	0.765	0.001
OS	12	BoSC	6	500	20	0.666	0.908	0.768	0.001
OS	24	BoSC	6	500	20	0.666	0.917	0.771	0.001
OS	6	BoSC	6	500	50	0.627	0.958	0.758	0.000
OS	12	BoSC	6	500	50	0.627	0.960	0.758	0.000
OS	24	BoSC	6	500	50	0.627	0.961	0.758	0.000
OS	6	BoSC	6	500	100	0.593	0.981	0.739	0.000
OS	12	BoSC	6	500	100	0.593	0.981	0.739	0.000
OS	24	BoSC	6	500	100	0.593	0.981	0.739	0.000
OS	6	BoSC	6	1000	5	0.890	0.611	0.725	0.005
OS	12	BoSC	6	1000	5	0.830	0.635	0.719	0.004
OS	24	BoSC	6	1000	5	0.776	0.684	0.727	0.003
OS	6	BoSC	6	1000	10	0.687	0.731	0.708	0.002
OS	12	BoSC	6	1000	10	0.687	0.749	0.717	0.002
OS	24	BoSC	6	1000	10	0.687	0.791	0.735	0.002
OS	6	BoSC	6	1000	20	0.666	0.821	0.736	0.001
OS	12	BoSC	6	1000	20	0.666	0.833	0.741	0.001
OS	24	BoSC	6	1000	20	0.666	0.858	0.750	0.001
OS	6	BoSC	6	1000	50	0.659	0.924	0.769	0.001
OS	12	BoSC	6	1000	50	0.659	0.931	0.772	0.000
OS	24	BoSC	6	1000	50	0.659	0.932	0.772	0.000
OS	6	BoSC	6	1000	100	0.593	0.959	0.733	0.000
OS	12	BoSC	6	1000	100	0.593	0.962	0.734	0.000
OS	24	BoSC	6	1000	100	0.593	0.963	0.734	0.000
OS	6	BoSC	6	5000	5	0.912	0.300	0.451	0.020
OS	12	BoSC	6	5000	5	0.853	0.340	0.487	0.016
OS	24	BoSC	6	5000	5	0.800	0.387	0.521	0.012
OS	6	BoSC	6	5000	10	0.690	0.431	0.531	0.009
OS	12	BoSC	6	5000	10	0.690	0.474	0.562	0.007
OS	24	BoSC	6	5000	10	0.690	0.540	0.606	0.006
OS	6	BoSC	6	5000	20	0.670	0.568	0.615	0.005
OS	12	BoSC	6	5000	20	0.670	0.586	0.625	0.005
OS	24	BoSC	6	5000	20	0.670	0.653	0.661	0.003
OS	6	BoSC	6	5000	50	0.670	0.717	0.692	0.003
OS	12	BoSC	6	5000	50	0.670	0.740	0.703	0.002
OS	24	BoSC	6	5000	50	0.670	0.778	0.720	0.002
OS	6	BoSC	6	5000	100	0.601	0.841	0.701	0.001
OS	12	BoSC	6	5000	100	0.601	0.856	0.706	0.001
OS	24	BoSC	6	5000	100	0.601	0.866	0.710	0.001
OS	6	STIDE	3	500	5	0.727	0.829	0.775	0.001
OS	12	STIDE	3	500	5	0.666	0.896	0.764	0.001
OS	24	STIDE	3	500	5	0.666	0.911	0.769	0.001
OS	6	STIDE	3	500	10	0.666	0.922	0.773	0.001
OS	12	STIDE	3	500	10	0.666	0.941	0.780	0.000
OS	24	STIDE	3	500	10	0.666	0.946	0.781	0.000
OS	6	STIDE	3	500	20	0.635	0.968	0.767	0.000

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
OS	12	STIDE	3	500	20	0.635	0.972	0.768	0.000
OS	24	STIDE	3	500	20	0.635	0.972	0.769	0.000
OS	6	STIDE	3	500	50	0.593	0.983	0.740	0.000
OS	12	STIDE	3	500	50	0.593	0.983	0.740	0.000
OS	24	STIDE	3	500	50	0.593	0.984	0.740	0.000
OS	6	STIDE	3	500	100	0.593	0.986	0.741	0.000
OS	12	STIDE	3	500	100	0.593	0.987	0.741	0.000
OS	24	STIDE	3	500	100	0.593	0.987	0.741	0.000
OS	6	STIDE	3	1000	5	0.728	0.749	0.738	0.002
OS	12	STIDE	3	1000	5	0.666	0.834	0.741	0.001
OS	24	STIDE	3	1000	5	0.666	0.861	0.751	0.001
OS	6	STIDE	3	1000	10	0.666	0.855	0.749	0.001
OS	12	STIDE	3	1000	10	0.666	0.901	0.766	0.001
OS	24	STIDE	3	1000	10	0.666	0.910	0.769	0.001
OS	6	STIDE	3	1000	20	0.659	0.935	0.773	0.000
OS	12	STIDE	3	1000	20	0.659	0.948	0.777	0.000
OS	24	STIDE	3	1000	20	0.659	0.950	0.778	0.000
OS	6	STIDE	3	1000	50	0.593	0.977	0.738	0.000
OS	12	STIDE	3	1000	50	0.593	0.978	0.738	0.000
OS	24	STIDE	3	1000	50	0.593	0.978	0.738	0.000
OS	6	STIDE	3	1000	100	0.593	0.982	0.739	0.000
OS	12	STIDE	3	1000	100	0.593	0.982	0.740	0.000
OS	24	STIDE	3	1000	100	0.593	0.983	0.740	0.000
OS	6	STIDE	3	5000	5	0.731	0.493	0.589	0.007
OS	12	STIDE	3	5000	5	0.670	0.586	0.625	0.005
OS	24	STIDE	3	5000	5	0.670	0.635	0.652	0.004
OS	6	STIDE	3	5000	10	0.670	0.601	0.633	0.004
OS	12	STIDE	3	5000	10	0.670	0.690	0.679	0.003
OS	24	STIDE	3	5000	10	0.670	0.727	0.697	0.002
OS	6	STIDE	3	5000	20	0.670	0.745	0.705	0.002
OS	12	STIDE	3	5000	20	0.670	0.815	0.735	0.001
OS	24	STIDE	3	5000	20	0.670	0.834	0.743	0.001
OS	6	STIDE	3	5000	50	0.594	0.900	0.715	0.001
OS	12	STIDE	3	5000	50	0.594	0.923	0.723	0.000
OS	24	STIDE	3	5000	50	0.594	0.927	0.724	0.000
OS	6	STIDE	3	5000	100	0.586	0.946	0.724	0.000
OS	12	STIDE	3	5000	100	0.586	0.951	0.725	0.000
OS	24	STIDE	3	5000	100	0.586	0.953	0.726	0.000
OS	6	STIDE	4	500	5	0.748	0.761	0.755	0.002
OS	12	STIDE	4	500	5	0.748	0.810	0.778	0.002
OS	24	STIDE	4	500	5	0.686	0.828	0.750	0.001
OS	6	STIDE	4	500	10	0.666	0.858	0.750	0.001
OS	12	STIDE	4	500	10	0.666	0.879	0.758	0.001
OS	24	STIDE	4	500	10	0.666	0.891	0.762	0.001
OS	6	STIDE	4	500	20	0.666	0.928	0.775	0.000
OS	12	STIDE	4	500	20	0.666	0.935	0.778	0.000
OS	24	STIDE	4	500	20	0.666	0.937	0.779	0.000
OS	6	STIDE	4	500	50	0.593	0.972	0.736	0.000
OS	12	STIDE	4	500	50	0.593	0.974	0.737	0.000
OS	24	STIDE	4	500	50	0.593	0.974	0.737	0.000
OS	6	STIDE	4	500	100	0.593	0.985	0.740	0.000
OS	12	STIDE	4	500	100	0.593	0.985	0.740	0.000
OS	24	STIDE	4	500	100	0.593	0.985	0.740	0.000
OS	6	STIDE	4	1000	5	0.748	0.667	0.705	0.003
OS	12	STIDE	4	1000	5	0.748	0.723	0.735	0.003
OS	24	STIDE	4	1000	5	0.687	0.752	0.718	0.002
OS	6	STIDE	4	1000	10	0.666	0.774	0.716	0.002
OS	12	STIDE	4	1000	10	0.666	0.809	0.731	0.001
OS	24	STIDE	4	1000	10	0.666	0.834	0.741	0.001
OS	6	STIDE	4	1000	20	0.666	0.859	0.751	0.001
OS	12	STIDE	4	1000	20	0.666	0.880	0.758	0.001
OS	24	STIDE	4	1000	20	0.666	0.888	0.762	0.001
OS	6	STIDE	4	1000	50	0.593	0.943	0.728	0.000

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
OS	12	STIDE	4	1000	50	0.593	0.946	0.729	0.000
OS	24	STIDE	4	1000	50	0.593	0.947	0.729	0.000
OS	6	STIDE	4	1000	100	0.593	0.976	0.738	0.000
OS	12	STIDE	4	1000	100	0.593	0.977	0.738	0.000
OS	24	STIDE	4	1000	100	0.593	0.978	0.738	0.000
OS	6	STIDE	4	5000	5	0.752	0.389	0.513	0.011
OS	12	STIDE	4	5000	5	0.752	0.448	0.562	0.009
OS	24	STIDE	4	5000	5	0.690	0.490	0.573	0.007
OS	6	STIDE	4	5000	10	0.670	0.516	0.583	0.006
OS	12	STIDE	4	5000	10	0.670	0.563	0.612	0.005
OS	24	STIDE	4	5000	10	0.670	0.621	0.644	0.004
OS	6	STIDE	4	5000	20	0.670	0.614	0.640	0.004
OS	12	STIDE	4	5000	20	0.670	0.663	0.666	0.003
OS	24	STIDE	4	5000	20	0.670	0.701	0.685	0.003
OS	6	STIDE	4	5000	50	0.609	0.768	0.679	0.002
OS	12	STIDE	4	5000	50	0.609	0.804	0.693	0.001
OS	24	STIDE	4	5000	50	0.609	0.817	0.698	0.001
OS	6	STIDE	4	5000	100	0.594	0.884	0.710	0.001
OS	12	STIDE	4	5000	100	0.594	0.898	0.715	0.001
OS	24	STIDE	4	5000	100	0.590	0.899	0.712	0.001
OS	6	STIDE	5	500	5	0.803	0.650	0.719	0.004
OS	12	STIDE	5	500	5	0.775	0.690	0.730	0.003
OS	24	STIDE	5	500	5	0.718	0.734	0.726	0.002
OS	6	STIDE	5	500	10	0.686	0.798	0.738	0.002
OS	12	STIDE	5	500	10	0.666	0.807	0.729	0.001
OS	24	STIDE	5	500	10	0.666	0.828	0.738	0.001
OS	6	STIDE	5	500	20	0.666	0.871	0.755	0.001
OS	12	STIDE	5	500	20	0.666	0.881	0.758	0.001
OS	24	STIDE	5	500	20	0.666	0.886	0.760	0.001
OS	6	STIDE	5	500	50	0.627	0.947	0.754	0.000
OS	12	STIDE	5	500	50	0.627	0.949	0.755	0.000
OS	24	STIDE	5	500	50	0.621	0.951	0.751	0.000
OS	6	STIDE	5	500	100	0.593	0.980	0.739	0.000
OS	12	STIDE	5	500	100	0.593	0.980	0.739	0.000
OS	24	STIDE	5	500	100	0.593	0.981	0.739	0.000
OS	6	STIDE	5	1000	5	0.803	0.515	0.627	0.007
OS	12	STIDE	5	1000	5	0.776	0.575	0.660	0.005
OS	24	STIDE	5	1000	5	0.719	0.631	0.672	0.004
OS	6	STIDE	5	1000	10	0.687	0.702	0.695	0.003
OS	12	STIDE	5	1000	10	0.666	0.718	0.691	0.002
OS	24	STIDE	5	1000	10	0.666	0.756	0.708	0.002
OS	6	STIDE	5	1000	20	0.666	0.795	0.725	0.002
OS	12	STIDE	5	1000	20	0.666	0.811	0.732	0.001
OS	24	STIDE	5	1000	20	0.666	0.827	0.738	0.001
OS	6	STIDE	5	1000	50	0.659	0.894	0.759	0.001
OS	12	STIDE	5	1000	50	0.659	0.902	0.762	0.001
OS	24	STIDE	5	1000	50	0.659	0.904	0.762	0.001
OS	6	STIDE	5	1000	100	0.593	0.951	0.731	0.000
OS	12	STIDE	5	1000	100	0.593	0.955	0.732	0.000
OS	24	STIDE	5	1000	100	0.593	0.957	0.732	0.000
OS	6	STIDE	5	5000	5	0.856	0.191	0.312	0.035
OS	12	STIDE	5	5000	5	0.799	0.266	0.399	0.021
OS	24	STIDE	5	5000	5	0.743	0.326	0.453	0.015
OS	6	STIDE	5	5000	10	0.690	0.401	0.507	0.010
OS	12	STIDE	5	5000	10	0.690	0.455	0.548	0.008
OS	24	STIDE	5	5000	10	0.690	0.519	0.592	0.006
OS	6	STIDE	5	5000	20	0.670	0.547	0.602	0.005
OS	12	STIDE	5	5000	20	0.670	0.580	0.622	0.005
OS	24	STIDE	5	5000	20	0.670	0.631	0.650	0.004
OS	6	STIDE	5	5000	50	0.670	0.691	0.680	0.003
OS	12	STIDE	5	5000	50	0.670	0.719	0.693	0.002
OS	24	STIDE	5	5000	50	0.670	0.745	0.705	0.002
OS	6	STIDE	5	5000	100	0.594	0.809	0.685	0.001

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
OS	12	STIDE	5	5000	100	0.594	0.822	0.689	0.001
OS	24	STIDE	5	5000	100	0.594	0.826	0.691	0.001
OS	6	STIDE	6	500	5	1.000	0.435	0.606	0.012
OS	12	STIDE	6	500	5	0.972	0.550	0.703	0.007
OS	24	STIDE	6	500	5	0.911	0.622	0.739	0.005
OS	6	STIDE	6	500	10	0.686	0.693	0.690	0.003
OS	12	STIDE	6	500	10	0.686	0.736	0.710	0.002
OS	24	STIDE	6	500	10	0.686	0.765	0.723	0.002
OS	6	STIDE	6	500	20	0.666	0.823	0.736	0.001
OS	12	STIDE	6	500	20	0.666	0.831	0.739	0.001
OS	24	STIDE	6	500	20	0.666	0.840	0.743	0.001
OS	6	STIDE	6	500	50	0.635	0.905	0.747	0.001
OS	12	STIDE	6	500	50	0.635	0.910	0.748	0.001
OS	24	STIDE	6	500	50	0.627	0.911	0.742	0.001
OS	6	STIDE	6	500	100	0.593	0.964	0.734	0.000
OS	12	STIDE	6	500	100	0.593	0.966	0.735	0.000
OS	24	STIDE	6	500	100	0.593	0.968	0.735	0.000
OS	6	STIDE	6	1000	5	1.000	0.269	0.424	0.025
OS	12	STIDE	6	1000	5	0.972	0.391	0.557	0.014
OS	24	STIDE	6	1000	5	0.911	0.477	0.626	0.009
OS	6	STIDE	6	1000	10	0.717	0.554	0.625	0.005
OS	12	STIDE	6	1000	10	0.687	0.623	0.654	0.004
OS	24	STIDE	6	1000	10	0.687	0.677	0.682	0.003
OS	6	STIDE	6	1000	20	0.666	0.733	0.698	0.002
OS	12	STIDE	6	1000	20	0.666	0.751	0.706	0.002
OS	24	STIDE	6	1000	20	0.666	0.773	0.716	0.002
OS	6	STIDE	6	1000	50	0.659	0.847	0.741	0.001
OS	12	STIDE	6	1000	50	0.659	0.852	0.743	0.001
OS	24	STIDE	6	1000	50	0.659	0.854	0.744	0.001
OS	6	STIDE	6	1000	100	0.593	0.904	0.716	0.001
OS	12	STIDE	6	1000	100	0.593	0.909	0.718	0.001
OS	24	STIDE	6	1000	100	0.593	0.911	0.718	0.001
OS	6	STIDE	6	5000	5	1.000	0.051	0.097	0.177
OS	12	STIDE	6	5000	5	1.000	0.100	0.182	0.085
OS	24	STIDE	6	5000	5	0.941	0.155	0.266	0.049
OS	6	STIDE	6	5000	10	0.828	0.156	0.262	0.043
OS	12	STIDE	6	5000	10	0.711	0.258	0.379	0.019
OS	24	STIDE	6	5000	10	0.711	0.355	0.474	0.012
OS	6	STIDE	6	5000	20	0.690	0.421	0.523	0.009
OS	12	STIDE	6	5000	20	0.690	0.500	0.580	0.007
OS	24	STIDE	6	5000	20	0.690	0.560	0.619	0.005
OS	6	STIDE	6	5000	50	0.670	0.624	0.646	0.004
OS	12	STIDE	6	5000	50	0.670	0.646	0.658	0.003
OS	24	STIDE	6	5000	50	0.670	0.673	0.671	0.003
OS	6	STIDE	6	5000	100	0.609	0.729	0.664	0.002
OS	12	STIDE	6	5000	100	0.609	0.744	0.670	0.002
OS	24	STIDE	6	5000	100	0.609	0.759	0.676	0.002
LXC	6	BoSC	3	500	5	1.000	0.619	0.765	0.005
LXC	12	BoSC	3	500	5	1.000	0.712	0.832	0.003
LXC	24	BoSC	3	500	5	0.999	0.820	0.901	0.002
LXC	6	BoSC	3	500	10	0.976	0.721	0.829	0.003
LXC	12	BoSC	3	500	10	0.969	0.839	0.899	0.002
LXC	24	BoSC	3	500	10	0.926	0.928	0.927	0.001
LXC	6	BoSC	3	500	20	0.902	0.861	0.881	0.001
LXC	12	BoSC	3	500	20	0.836	0.949	0.889	0.000
LXC	24	BoSC	3	500	20	0.820	0.985	0.895	0.000
LXC	6	BoSC	3	500	50	0.820	0.985	0.895	0.000
LXC	12	BoSC	3	500	50	0.820	0.995	0.899	0.000
LXC	24	BoSC	3	500	50	0.820	0.998	0.900	0.000
LXC	6	BoSC	3	500	100	0.820	0.996	0.899	0.000
LXC	12	BoSC	3	500	100	0.810	0.999	0.895	0.000
LXC	24	BoSC	3	500	100	0.603	0.999	0.752	0.000
LXC	6	BoSC	3	1000	5	1.000	0.523	0.687	0.007

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
LXC	12	BoSC	3	1000	5	1.000	0.595	0.746	0.006
LXC	24	BoSC	3	1000	5	0.999	0.701	0.824	0.003
LXC	6	BoSC	3	1000	10	0.978	0.596	0.741	0.005
LXC	12	BoSC	3	1000	10	0.972	0.707	0.819	0.003
LXC	24	BoSC	3	1000	10	0.938	0.832	0.882	0.002
LXC	6	BoSC	3	1000	20	0.953	0.721	0.821	0.003
LXC	12	BoSC	3	1000	20	0.876	0.851	0.863	0.001
LXC	24	BoSC	3	1000	20	0.835	0.948	0.888	0.000
LXC	6	BoSC	3	1000	50	0.838	0.930	0.882	0.001
LXC	12	BoSC	3	1000	50	0.819	0.984	0.894	0.000
LXC	24	BoSC	3	1000	50	0.819	0.996	0.899	0.000
LXC	6	BoSC	3	1000	100	0.819	0.990	0.897	0.000
LXC	12	BoSC	3	1000	100	0.819	0.997	0.900	0.000
LXC	24	BoSC	3	1000	100	0.819	0.999	0.900	0.000
LXC	6	BoSC	3	5000	5	1.000	0.298	0.460	0.019
LXC	12	BoSC	3	5000	5	1.000	0.332	0.498	0.016
LXC	24	BoSC	3	5000	5	0.999	0.393	0.564	0.013
LXC	6	BoSC	3	5000	10	0.978	0.334	0.498	0.016
LXC	12	BoSC	3	5000	10	0.978	0.388	0.556	0.013
LXC	24	BoSC	3	5000	10	0.975	0.492	0.654	0.008
LXC	6	BoSC	3	5000	20	0.965	0.391	0.557	0.012
LXC	12	BoSC	3	5000	20	0.959	0.494	0.652	0.008
LXC	24	BoSC	3	5000	20	0.914	0.664	0.769	0.004
LXC	6	BoSC	3	5000	50	0.939	0.558	0.700	0.006
LXC	12	BoSC	3	5000	50	0.824	0.758	0.790	0.002
LXC	24	BoSC	3	5000	50	0.819	0.928	0.870	0.001
LXC	6	BoSC	3	5000	100	0.824	0.807	0.816	0.002
LXC	12	BoSC	3	5000	100	0.819	0.951	0.880	0.000
LXC	24	BoSC	3	5000	100	0.819	0.991	0.897	0.000
LXC	6	BoSC	4	500	5	1.000	0.508	0.673	0.007
LXC	12	BoSC	4	500	5	1.000	0.546	0.706	0.006
LXC	24	BoSC	4	500	5	1.000	0.603	0.753	0.005
LXC	6	BoSC	4	500	10	1.000	0.556	0.715	0.006
LXC	12	BoSC	4	500	10	1.000	0.616	0.763	0.005
LXC	24	BoSC	4	500	10	1.000	0.697	0.822	0.003
LXC	6	BoSC	4	500	20	0.975	0.634	0.768	0.004
LXC	12	BoSC	4	500	20	0.974	0.723	0.830	0.003
LXC	24	BoSC	4	500	20	0.964	0.825	0.889	0.002
LXC	6	BoSC	4	500	50	0.950	0.825	0.883	0.002
LXC	12	BoSC	4	500	50	0.937	0.924	0.930	0.001
LXC	24	BoSC	4	500	50	0.931	0.977	0.953	0.000
LXC	6	BoSC	4	500	100	0.870	0.970	0.917	0.000
LXC	12	BoSC	4	500	100	0.809	0.993	0.891	0.000
LXC	24	BoSC	4	500	100	0.809	0.997	0.893	0.000
LXC	6	BoSC	4	1000	5	1.000	0.441	0.613	0.010
LXC	12	BoSC	4	1000	5	1.000	0.467	0.637	0.009
LXC	24	BoSC	4	1000	5	1.000	0.510	0.676	0.007
LXC	6	BoSC	4	1000	10	1.000	0.474	0.644	0.008
LXC	12	BoSC	4	1000	10	1.000	0.516	0.681	0.007
LXC	24	BoSC	4	1000	10	1.000	0.580	0.734	0.006
LXC	6	BoSC	4	1000	20	0.977	0.525	0.683	0.007
LXC	12	BoSC	4	1000	20	0.977	0.593	0.738	0.005
LXC	24	BoSC	4	1000	20	0.964	0.685	0.801	0.003
LXC	6	BoSC	4	1000	50	0.963	0.666	0.788	0.004
LXC	12	BoSC	4	1000	50	0.950	0.784	0.859	0.002
LXC	24	BoSC	4	1000	50	0.936	0.899	0.917	0.001
LXC	6	BoSC	4	1000	100	0.936	0.850	0.891	0.001
LXC	12	BoSC	4	1000	100	0.936	0.950	0.943	0.000
LXC	24	BoSC	4	1000	100	0.862	0.985	0.920	0.000
LXC	6	BoSC	4	5000	5	1.000	0.250	0.400	0.023
LXC	12	BoSC	4	5000	5	1.000	0.264	0.418	0.021
LXC	24	BoSC	4	5000	5	1.000	0.288	0.447	0.019
LXC	6	BoSC	4	5000	10	1.000	0.269	0.425	0.021

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
LXC	12	BoSC	4	5000	10	1.000	0.292	0.452	0.019
LXC	24	BoSC	4	5000	10	1.000	0.325	0.491	0.016
LXC	6	BoSC	4	5000	20	0.977	0.298	0.456	0.018
LXC	12	BoSC	4	5000	20	0.977	0.329	0.492	0.015
LXC	24	BoSC	4	5000	20	0.963	0.371	0.535	0.013
LXC	6	BoSC	4	5000	50	0.963	0.361	0.525	0.013
LXC	12	BoSC	4	5000	50	0.963	0.414	0.579	0.010
LXC	24	BoSC	4	5000	50	0.949	0.515	0.668	0.007
LXC	6	BoSC	4	5000	100	0.963	0.441	0.605	0.009
LXC	12	BoSC	4	5000	100	0.949	0.571	0.713	0.005
LXC	24	BoSC	4	5000	100	0.936	0.763	0.841	0.002
LXC	6	BoSC	5	500	5	1.000	0.479	0.648	0.008
LXC	12	BoSC	5	500	5	1.000	0.498	0.665	0.008
LXC	24	BoSC	5	500	5	1.000	0.531	0.694	0.007
LXC	6	BoSC	5	500	10	1.000	0.507	0.673	0.007
LXC	12	BoSC	5	500	10	1.000	0.539	0.700	0.007
LXC	24	BoSC	5	500	10	1.000	0.587	0.740	0.005
LXC	6	BoSC	5	500	20	0.977	0.550	0.704	0.006
LXC	12	BoSC	5	500	20	0.977	0.601	0.744	0.005
LXC	24	BoSC	5	500	20	0.977	0.672	0.796	0.004
LXC	6	BoSC	5	500	50	0.963	0.680	0.797	0.003
LXC	12	BoSC	5	500	50	0.963	0.765	0.853	0.002
LXC	24	BoSC	5	500	50	0.950	0.857	0.901	0.001
LXC	6	BoSC	5	500	100	0.937	0.848	0.890	0.001
LXC	12	BoSC	5	500	100	0.937	0.932	0.934	0.001
LXC	24	BoSC	5	500	100	0.876	0.978	0.924	0.000
LXC	6	BoSC	5	1000	5	1.000	0.421	0.593	0.010
LXC	12	BoSC	5	1000	5	1.000	0.435	0.606	0.010
LXC	24	BoSC	5	1000	5	1.000	0.458	0.628	0.009
LXC	6	BoSC	5	1000	10	1.000	0.443	0.614	0.010
LXC	12	BoSC	5	1000	10	1.000	0.463	0.633	0.009
LXC	24	BoSC	5	1000	10	1.000	0.497	0.664	0.008
LXC	6	BoSC	5	1000	20	0.977	0.469	0.634	0.008
LXC	12	BoSC	5	1000	20	0.977	0.504	0.665	0.007
LXC	24	BoSC	5	1000	20	0.977	0.556	0.709	0.006
LXC	6	BoSC	5	1000	50	0.964	0.553	0.703	0.006
LXC	12	BoSC	5	1000	50	0.964	0.619	0.754	0.005
LXC	24	BoSC	5	1000	50	0.963	0.711	0.818	0.003
LXC	6	BoSC	5	1000	100	0.963	0.680	0.797	0.003
LXC	12	BoSC	5	1000	100	0.936	0.778	0.850	0.002
LXC	24	BoSC	5	1000	100	0.936	0.885	0.910	0.001
LXC	6	BoSC	5	5000	5	1.000	0.235	0.381	0.025
LXC	12	BoSC	5	5000	5	1.000	0.244	0.392	0.024
LXC	24	BoSC	5	5000	5	1.000	0.258	0.411	0.022
LXC	6	BoSC	5	5000	10	1.000	0.251	0.401	0.023
LXC	12	BoSC	5	5000	10	1.000	0.263	0.416	0.022
LXC	24	BoSC	5	5000	10	1.000	0.283	0.442	0.019
LXC	6	BoSC	5	5000	20	0.983	0.268	0.421	0.021
LXC	12	BoSC	5	5000	20	0.977	0.286	0.443	0.019
LXC	24	BoSC	5	5000	20	0.977	0.314	0.475	0.016
LXC	6	BoSC	5	5000	50	0.963	0.315	0.475	0.016
LXC	12	BoSC	5	5000	50	0.963	0.344	0.506	0.014
LXC	24	BoSC	5	5000	50	0.963	0.381	0.546	0.012
LXC	6	BoSC	5	5000	100	0.963	0.367	0.532	0.013
LXC	12	BoSC	5	5000	100	0.963	0.408	0.573	0.011
LXC	24	BoSC	5	5000	100	0.963	0.487	0.646	0.008
LXC	6	BoSC	6	500	5	1.000	0.463	0.633	0.009
LXC	12	BoSC	6	500	5	1.000	0.476	0.645	0.008
LXC	24	BoSC	6	500	5	1.000	0.498	0.664	0.008
LXC	6	BoSC	6	500	10	1.000	0.487	0.655	0.008
LXC	12	BoSC	6	500	10	1.000	0.505	0.671	0.007
LXC	24	BoSC	6	500	10	1.000	0.537	0.699	0.007
LXC	6	BoSC	6	500	20	0.994	0.517	0.680	0.007

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
LXC	12	BoSC	6	500	20	0.983	0.547	0.703	0.006
LXC	24	BoSC	6	500	20	0.977	0.594	0.739	0.005
LXC	6	BoSC	6	500	50	0.965	0.602	0.741	0.005
LXC	12	BoSC	6	500	50	0.964	0.664	0.786	0.004
LXC	24	BoSC	6	500	50	0.963	0.742	0.838	0.003
LXC	6	BoSC	6	500	100	0.963	0.744	0.839	0.003
LXC	12	BoSC	6	500	100	0.950	0.823	0.882	0.002
LXC	24	BoSC	6	500	100	0.937	0.902	0.919	0.001
LXC	6	BoSC	6	1000	5	1.000	0.406	0.577	0.011
LXC	12	BoSC	6	1000	5	1.000	0.417	0.589	0.011
LXC	24	BoSC	6	1000	5	1.000	0.433	0.604	0.010
LXC	6	BoSC	6	1000	10	1.000	0.429	0.600	0.010
LXC	12	BoSC	6	1000	10	1.000	0.441	0.612	0.010
LXC	24	BoSC	6	1000	10	1.000	0.463	0.633	0.009
LXC	6	BoSC	6	1000	20	1.000	0.451	0.622	0.009
LXC	12	BoSC	6	1000	20	1.000	0.472	0.641	0.009
LXC	24	BoSC	6	1000	20	1.000	0.506	0.672	0.007
LXC	6	BoSC	6	1000	50	0.966	0.503	0.661	0.007
LXC	12	BoSC	6	1000	50	0.966	0.544	0.696	0.006
LXC	24	BoSC	6	1000	50	0.965	0.605	0.743	0.005
LXC	6	BoSC	6	1000	100	0.963	0.591	0.732	0.005
LXC	12	BoSC	6	1000	100	0.963	0.661	0.784	0.004
LXC	24	BoSC	6	1000	100	0.963	0.753	0.845	0.002
LXC	6	BoSC	6	5000	5	1.000	0.216	0.356	0.028
LXC	12	BoSC	6	5000	5	1.000	0.227	0.370	0.026
LXC	24	BoSC	6	5000	5	1.000	0.239	0.386	0.024
LXC	6	BoSC	6	5000	10	1.000	0.240	0.387	0.024
LXC	12	BoSC	6	5000	10	1.000	0.248	0.398	0.023
LXC	24	BoSC	6	5000	10	1.000	0.262	0.415	0.022
LXC	6	BoSC	6	5000	20	1.000	0.256	0.408	0.022
LXC	12	BoSC	6	5000	20	1.000	0.269	0.424	0.021
LXC	24	BoSC	6	5000	20	1.000	0.290	0.449	0.019
LXC	6	BoSC	6	5000	50	0.966	0.288	0.444	0.018
LXC	12	BoSC	6	5000	50	0.966	0.310	0.469	0.017
LXC	24	BoSC	6	5000	50	0.964	0.338	0.500	0.014
LXC	6	BoSC	6	5000	100	0.963	0.335	0.498	0.015
LXC	12	BoSC	6	5000	100	0.963	0.361	0.525	0.013
LXC	24	BoSC	6	5000	100	0.963	0.397	0.562	0.011
LXC	6	STIDE	3	500	5	1.000	0.581	0.735	0.006
LXC	12	STIDE	3	500	5	1.000	0.651	0.789	0.005
LXC	24	STIDE	3	500	5	0.999	0.745	0.854	0.003
LXC	6	STIDE	3	500	10	0.986	0.658	0.789	0.004
LXC	12	STIDE	3	500	10	0.980	0.758	0.855	0.003
LXC	24	STIDE	3	500	10	0.929	0.862	0.894	0.001
LXC	6	STIDE	3	500	20	0.967	0.783	0.865	0.002
LXC	12	STIDE	3	500	20	0.882	0.890	0.886	0.001
LXC	24	STIDE	3	500	20	0.833	0.963	0.893	0.000
LXC	6	STIDE	3	500	50	0.830	0.962	0.891	0.000
LXC	12	STIDE	3	500	50	0.828	0.991	0.902	0.000
LXC	24	STIDE	3	500	50	0.828	0.997	0.904	0.000
LXC	6	STIDE	3	500	100	0.828	0.995	0.904	0.000
LXC	12	STIDE	3	500	100	0.818	0.998	0.899	0.000
LXC	24	STIDE	3	500	100	0.776	0.999	0.874	0.000
LXC	6	STIDE	3	1000	5	1.000	0.499	0.666	0.008
LXC	12	STIDE	3	1000	5	1.000	0.550	0.709	0.007
LXC	24	STIDE	3	1000	5	0.999	0.629	0.772	0.005
LXC	6	STIDE	3	1000	10	0.988	0.552	0.708	0.007
LXC	12	STIDE	3	1000	10	0.988	0.633	0.772	0.005
LXC	24	STIDE	3	1000	10	0.964	0.743	0.839	0.003
LXC	6	STIDE	3	1000	20	0.967	0.643	0.773	0.005
LXC	12	STIDE	3	1000	20	0.945	0.763	0.844	0.002
LXC	24	STIDE	3	1000	20	0.878	0.885	0.881	0.001
LXC	6	STIDE	3	1000	50	0.949	0.856	0.900	0.001

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
LXC	12	STIDE	3	1000	50	0.830	0.953	0.887	0.000
LXC	24	STIDE	3	1000	50	0.827	0.989	0.901	0.000
LXC	6	STIDE	3	1000	100	0.827	0.979	0.897	0.000
LXC	12	STIDE	3	1000	100	0.827	0.995	0.903	0.000
LXC	24	STIDE	3	1000	100	0.827	0.998	0.905	0.000
LXC	6	STIDE	3	5000	5	1.000	0.289	0.448	0.021
LXC	12	STIDE	3	5000	5	1.000	0.314	0.478	0.019
LXC	24	STIDE	3	5000	5	0.999	0.356	0.525	0.015
LXC	6	STIDE	3	5000	10	0.988	0.317	0.480	0.018
LXC	12	STIDE	3	5000	10	0.988	0.356	0.523	0.015
LXC	24	STIDE	3	5000	10	0.976	0.418	0.585	0.012
LXC	6	STIDE	3	5000	20	0.967	0.362	0.526	0.015
LXC	12	STIDE	3	5000	20	0.961	0.420	0.584	0.011
LXC	24	STIDE	3	5000	20	0.942	0.533	0.681	0.007
LXC	6	STIDE	3	5000	50	0.966	0.467	0.630	0.009
LXC	12	STIDE	3	5000	50	0.899	0.618	0.732	0.005
LXC	24	STIDE	3	5000	50	0.832	0.816	0.824	0.002
LXC	6	STIDE	3	5000	100	0.942	0.668	0.782	0.004
LXC	12	STIDE	3	5000	100	0.829	0.870	0.849	0.001
LXC	24	STIDE	3	5000	100	0.827	0.973	0.894	0.000
LXC	6	STIDE	4	500	5	1.000	0.507	0.672	0.008
LXC	12	STIDE	4	500	5	1.000	0.525	0.689	0.008
LXC	24	STIDE	4	500	5	1.000	0.558	0.716	0.007
LXC	6	STIDE	4	500	10	1.000	0.534	0.696	0.007
LXC	12	STIDE	4	500	10	1.000	0.566	0.723	0.007
LXC	24	STIDE	4	500	10	1.000	0.613	0.760	0.005
LXC	6	STIDE	4	500	20	0.978	0.578	0.726	0.006
LXC	12	STIDE	4	500	20	0.978	0.630	0.767	0.005
LXC	24	STIDE	4	500	20	0.977	0.699	0.815	0.004
LXC	6	STIDE	4	500	50	0.967	0.711	0.819	0.003
LXC	12	STIDE	4	500	50	0.955	0.792	0.866	0.002
LXC	24	STIDE	4	500	50	0.943	0.875	0.908	0.001
LXC	6	STIDE	4	500	100	0.943	0.873	0.907	0.001
LXC	12	STIDE	4	500	100	0.943	0.945	0.944	0.000
LXC	24	STIDE	4	500	100	0.926	0.984	0.954	0.000
LXC	6	STIDE	4	1000	5	1.000	0.449	0.619	0.010
LXC	12	STIDE	4	1000	5	1.000	0.461	0.631	0.010
LXC	24	STIDE	4	1000	5	1.000	0.484	0.652	0.009
LXC	6	STIDE	4	1000	10	1.000	0.469	0.639	0.010
LXC	12	STIDE	4	1000	10	1.000	0.489	0.657	0.009
LXC	24	STIDE	4	1000	10	1.000	0.523	0.687	0.008
LXC	6	STIDE	4	1000	20	0.979	0.495	0.658	0.008
LXC	12	STIDE	4	1000	20	0.979	0.531	0.689	0.007
LXC	24	STIDE	4	1000	20	0.977	0.584	0.731	0.006
LXC	6	STIDE	4	1000	50	0.967	0.584	0.728	0.006
LXC	12	STIDE	4	1000	50	0.967	0.651	0.778	0.004
LXC	24	STIDE	4	1000	50	0.967	0.740	0.838	0.003
LXC	6	STIDE	4	1000	100	0.967	0.713	0.821	0.003
LXC	12	STIDE	4	1000	100	0.955	0.811	0.877	0.002
LXC	24	STIDE	4	1000	100	0.943	0.903	0.922	0.001
LXC	6	STIDE	4	5000	5	1.000	0.256	0.408	0.025
LXC	12	STIDE	4	5000	5	1.000	0.265	0.419	0.024
LXC	24	STIDE	4	5000	5	1.000	0.278	0.435	0.022
LXC	6	STIDE	4	5000	10	1.000	0.271	0.427	0.023
LXC	12	STIDE	4	5000	10	1.000	0.282	0.440	0.022
LXC	24	STIDE	4	5000	10	1.000	0.303	0.465	0.020
LXC	6	STIDE	4	5000	20	0.979	0.286	0.442	0.021
LXC	12	STIDE	4	5000	20	0.979	0.307	0.468	0.019
LXC	24	STIDE	4	5000	20	0.977	0.337	0.501	0.016
LXC	6	STIDE	4	5000	50	0.966	0.339	0.502	0.016
LXC	12	STIDE	4	5000	50	0.966	0.370	0.535	0.014
LXC	24	STIDE	4	5000	50	0.966	0.409	0.575	0.012
LXC	6	STIDE	4	5000	100	0.966	0.396	0.562	0.013

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
LXC	12	STIDE	4	5000	100	0.966	0.438	0.603	0.011
LXC	24	STIDE	4	5000	100	0.966	0.517	0.673	0.008
LXC	6	STIDE	5	500	5	1.000	0.465	0.635	0.010
LXC	12	STIDE	5	500	5	1.000	0.483	0.652	0.009
LXC	24	STIDE	5	500	5	1.000	0.505	0.671	0.008
LXC	6	STIDE	5	500	10	1.000	0.503	0.669	0.008
LXC	12	STIDE	5	500	10	1.000	0.515	0.680	0.008
LXC	24	STIDE	5	500	10	1.000	0.536	0.698	0.007
LXC	6	STIDE	5	500	20	1.000	0.527	0.690	0.008
LXC	12	STIDE	5	500	20	0.988	0.545	0.703	0.007
LXC	24	STIDE	5	500	20	0.979	0.577	0.726	0.006
LXC	6	STIDE	5	500	50	0.967	0.589	0.732	0.006
LXC	12	STIDE	5	500	50	0.967	0.636	0.767	0.005
LXC	24	STIDE	5	500	50	0.967	0.697	0.810	0.004
LXC	6	STIDE	5	500	100	0.967	0.711	0.819	0.003
LXC	12	STIDE	5	500	100	0.967	0.776	0.861	0.002
LXC	24	STIDE	5	500	100	0.943	0.840	0.889	0.002
LXC	6	STIDE	5	1000	5	1.000	0.393	0.564	0.013
LXC	12	STIDE	5	1000	5	1.000	0.417	0.589	0.012
LXC	24	STIDE	5	1000	5	1.000	0.443	0.614	0.011
LXC	6	STIDE	5	1000	10	1.000	0.446	0.617	0.011
LXC	12	STIDE	5	1000	10	1.000	0.456	0.627	0.010
LXC	24	STIDE	5	1000	10	1.000	0.470	0.640	0.010
LXC	6	STIDE	5	1000	20	1.000	0.466	0.636	0.010
LXC	12	STIDE	5	1000	20	1.000	0.479	0.648	0.009
LXC	24	STIDE	5	1000	20	0.979	0.495	0.658	0.008
LXC	6	STIDE	5	1000	50	0.969	0.502	0.661	0.008
LXC	12	STIDE	5	1000	50	0.967	0.532	0.687	0.007
LXC	24	STIDE	5	1000	50	0.967	0.577	0.723	0.006
LXC	6	STIDE	5	1000	100	0.967	0.577	0.723	0.006
LXC	12	STIDE	5	1000	100	0.967	0.627	0.761	0.005
LXC	24	STIDE	5	1000	100	0.967	0.693	0.807	0.004
LXC	6	STIDE	5	5000	5	1.000	0.154	0.266	0.047
LXC	12	STIDE	5	5000	5	1.000	0.192	0.322	0.036
LXC	24	STIDE	5	5000	5	1.000	0.240	0.387	0.027
LXC	6	STIDE	5	5000	10	1.000	0.245	0.393	0.026
LXC	12	STIDE	5	5000	10	1.000	0.259	0.411	0.024
LXC	24	STIDE	5	5000	10	1.000	0.271	0.426	0.023
LXC	6	STIDE	5	5000	20	1.000	0.270	0.425	0.023
LXC	12	STIDE	5	5000	20	1.000	0.277	0.434	0.022
LXC	24	STIDE	5	5000	20	0.979	0.285	0.442	0.021
LXC	6	STIDE	5	5000	50	0.969	0.290	0.446	0.020
LXC	12	STIDE	5	5000	50	0.967	0.309	0.469	0.018
LXC	24	STIDE	5	5000	50	0.967	0.335	0.498	0.016
LXC	6	STIDE	5	5000	100	0.966	0.339	0.502	0.016
LXC	12	STIDE	5	5000	100	0.966	0.363	0.528	0.014
LXC	24	STIDE	5	5000	100	0.966	0.390	0.555	0.013
LXC	6	STIDE	6	500	5	1.000	0.279	0.437	0.022
LXC	12	STIDE	6	500	5	1.000	0.335	0.502	0.017
LXC	24	STIDE	6	500	5	1.000	0.431	0.602	0.011
LXC	6	STIDE	6	500	10	1.000	0.448	0.619	0.010
LXC	12	STIDE	6	500	10	1.000	0.474	0.643	0.009
LXC	24	STIDE	6	500	10	1.000	0.505	0.671	0.008
LXC	6	STIDE	6	500	20	1.000	0.505	0.671	0.008
LXC	12	STIDE	6	500	20	1.000	0.517	0.682	0.008
LXC	24	STIDE	6	500	20	1.000	0.535	0.697	0.007
LXC	6	STIDE	6	500	50	0.969	0.542	0.695	0.007
LXC	12	STIDE	6	500	50	0.969	0.566	0.715	0.006
LXC	24	STIDE	6	500	50	0.967	0.604	0.743	0.005
LXC	6	STIDE	6	500	100	0.967	0.622	0.757	0.005
LXC	12	STIDE	6	500	100	0.967	0.668	0.790	0.004
LXC	24	STIDE	6	500	100	0.967	0.724	0.828	0.003
LXC	6	STIDE	6	1000	5	1.000	0.168	0.287	0.042

Table A.1: An overview of the results for all platforms.

Platform	Train Time (H)	Algo	Window/ Dec. Thr.	Epoch Size	Det. Thr.	Recall	Prec.	F-Meas.	FPR
LXC	12	STIDE	6	1000	5	1.000	0.219	0.360	0.030
LXC	24	STIDE	6	1000	5	1.000	0.335	0.502	0.017
LXC	6	STIDE	6	1000	10	1.000	0.343	0.511	0.016
LXC	12	STIDE	6	1000	10	1.000	0.388	0.560	0.013
LXC	24	STIDE	6	1000	10	1.000	0.442	0.613	0.011
LXC	6	STIDE	6	1000	20	1.000	0.444	0.615	0.011
LXC	12	STIDE	6	1000	20	1.000	0.457	0.627	0.010
LXC	24	STIDE	6	1000	20	1.000	0.470	0.640	0.010
LXC	6	STIDE	6	1000	50	0.970	0.473	0.636	0.009
LXC	12	STIDE	6	1000	50	0.970	0.488	0.649	0.009
LXC	24	STIDE	6	1000	50	0.970	0.512	0.670	0.008
LXC	6	STIDE	6	1000	100	0.967	0.520	0.676	0.008
LXC	12	STIDE	6	1000	100	0.967	0.549	0.701	0.007
LXC	24	STIDE	6	1000	100	0.967	0.590	0.733	0.006
LXC	6	STIDE	6	5000	5	1.000	0.037	0.072	0.219
LXC	12	STIDE	6	5000	5	1.000	0.050	0.095	0.163
LXC	24	STIDE	6	5000	5	1.000	0.100	0.181	0.077
LXC	6	STIDE	6	5000	10	1.000	0.076	0.140	0.104
LXC	12	STIDE	6	5000	10	1.000	0.108	0.195	0.070
LXC	24	STIDE	6	5000	10	1.000	0.206	0.342	0.033
LXC	6	STIDE	6	5000	20	1.000	0.184	0.310	0.038
LXC	12	STIDE	6	5000	20	1.000	0.229	0.372	0.029
LXC	24	STIDE	6	5000	20	1.000	0.269	0.423	0.023
LXC	6	STIDE	6	5000	50	0.976	0.274	0.427	0.022
LXC	12	STIDE	6	5000	50	0.970	0.280	0.435	0.021
LXC	24	STIDE	6	5000	50	0.970	0.296	0.453	0.020
LXC	6	STIDE	6	5000	100	0.966	0.302	0.460	0.019
LXC	12	STIDE	6	5000	100	0.966	0.320	0.481	0.017
LXC	24	STIDE	6	5000	100	0.966	0.344	0.507	0.016

This page is intentionally left blank.

Appendix B

Complete Experimental Results for Expected Cost Analysis for all Platforms

Below we list the complete expected cost result obtained from the analysis.

Table B.1: An overview of the results of expected cost for all platforms.

Platform	Algo	Epoch Size	Det. Thr.	P(I)	Recall	FPR.	Expected Cost
Docker	BoSC	500	5	0.024	1.000	0.008	0.0076
Docker	BoSC	500	10	0.024	1.000	0.005	0.0053
Docker	BoSC	500	20	0.024	0.988	0.004	0.0066
Docker	BoSC	500	50	0.024	0.975	0.002	0.0083
Docker	BoSC	500	100	0.024	0.975	0.002	0.0082
Docker	BoSC	1000	5	0.024	1.000	0.012	0.0117
Docker	BoSC	1000	10	0.024	1.000	0.008	0.0082
Docker	BoSC	1000	20	0.024	0.989	0.006	0.0082
Docker	BoSC	1000	50	0.024	0.975	0.003	0.0088
Docker	BoSC	1000	100	0.024	0.975	0.002	0.0085
Docker	BoSC	5000	5	0.025	1.000	0.032	0.0309
Docker	BoSC	5000	10	0.025	1.000	0.024	0.0238
Docker	BoSC	5000	20	0.025	1.000	0.018	0.0180
Docker	BoSC	5000	50	0.025	0.971	0.006	0.0126
Docker	BoSC	5000	100	0.025	0.970	0.004	0.0112
Docker	STIDE	500	5	0.024	1.000	0.008	0.0081
Docker	STIDE	500	10	0.024	1.000	0.006	0.0057
Docker	STIDE	500	20	0.024	0.993	0.004	0.0058
Docker	STIDE	500	50	0.024	0.975	0.003	0.0085
Docker	STIDE	500	100	0.024	0.975	0.002	0.0082
Docker	STIDE	1000	5	0.024	1.000	0.013	0.0124
Docker	STIDE	1000	10	0.024	1.000	0.009	0.0087
Docker	STIDE	1000	20	0.024	0.993	0.006	0.0078

Table B.1: An overview of the results of expected cost for all platforms.

Platform	Algo	Epoch Size	Det. Thr.	P(I)	Recall	FPR.	Expected Cost
Docker	STIDE	1000	50	0.024	0.975	0.003	0.0092
Docker	STIDE	1000	100	0.024	0.975	0.003	0.0086
Docker	STIDE	5000	5	0.025	1.000	0.034	0.0331
Docker	STIDE	5000	10	0.025	1.000	0.025	0.0248
Docker	STIDE	5000	20	0.025	1.000	0.020	0.0192
Docker	STIDE	5000	50	0.025	0.973	0.007	0.0134
Docker	STIDE	5000	100	0.025	0.970	0.004	0.0118
Docker	HMM	500	5	0.026	0.895	0.012	0.0383
Docker	HMM	500	10	0.026	0.895	0.005	0.0318
Docker	HMM	500	20	0.026	0.880	0.003	0.0340
Docker	HMM	500	50	0.026	0.824	0.002	0.0475
Docker	HMM	500	100	0.026	0.795	0.002	0.0547
Docker	HMM	1000	5	0.027	0.890	0.020	0.0491
Docker	HMM	1000	10	0.027	0.890	0.013	0.0427
Docker	HMM	1000	20	0.027	0.876	0.005	0.0389
Docker	HMM	1000	50	0.027	0.876	0.003	0.0369
Docker	HMM	1000	100	0.027	0.807	0.003	0.0551
Docker	HMM	5000	5	0.035	0.907	0.046	0.0768
Docker	HMM	5000	10	0.035	0.907	0.036	0.0670
Docker	HMM	5000	20	0.035	0.907	0.033	0.0647
Docker	HMM	5000	50	0.035	0.900	0.024	0.0579
Docker	HMM	5000	100	0.035	0.867	0.014	0.0599
LXC	BoSC	500	5	0.008	1.000	0.003	0.0034
LXC	BoSC	500	10	0.008	0.962	0.002	0.0049
LXC	BoSC	500	20	0.008	0.890	0.001	0.0094
LXC	BoSC	500	50	0.008	0.873	0.000	0.0100
LXC	BoSC	500	100	0.008	0.703	0.000	0.0232
LXC	BoSC	1000	5	0.008	1.000	0.005	0.0054
LXC	BoSC	1000	10	0.008	0.968	0.004	0.0060
LXC	BoSC	1000	20	0.008	0.898	0.002	0.0099
LXC	BoSC	1000	50	0.008	0.876	0.000	0.0101
LXC	BoSC	1000	100	0.008	0.840	0.000	0.0125
LXC	BoSC	5000	5	0.008	0.999	0.016	0.0157
LXC	BoSC	5000	10	0.008	0.987	0.012	0.0130
LXC	BoSC	5000	20	0.008	0.938	0.008	0.0129
LXC	BoSC	5000	50	0.008	0.882	0.004	0.0129
LXC	BoSC	5000	100	0.008	0.876	0.001	0.0109
LXC	STIDE	500	5	0.008	1.000	0.005	0.0048
LXC	STIDE	500	10	0.008	0.965	0.003	0.0063
LXC	STIDE	500	20	0.008	0.905	0.002	0.0099
LXC	STIDE	500	50	0.008	0.885	0.001	0.0102
LXC	STIDE	500	100	0.008	0.851	0.000	0.0126
LXC	STIDE	1000	5	0.008	1.000	0.007	0.0070

Table B.1: An overview of the results of expected cost for all platforms.

Platform	**Algo**	**Epoch Size**	**Det. Thr.**	**P(I)**	**Recall**	**FPR.**	**Expected Cost**
LXC	STIDE	1000	10	0.008	0.982	0.005	0.0067
LXC	STIDE	1000	20	0.008	0.928	0.003	0.0095
LXC	STIDE	1000	50	0.008	0.897	0.001	0.0101
LXC	STIDE	1000	100	0.008	0.885	0.000	0.0101
LXC	STIDE	5000	5	0.008	1.000	0.019	0.0188
LXC	STIDE	5000	10	0.008	0.988	0.016	0.0168
LXC	STIDE	5000	20	0.008	0.959	0.012	0.0152
LXC	STIDE	5000	50	0.008	0.899	0.007	0.0150
LXC	STIDE	5000	100	0.008	0.896	0.004	0.0123
OS	BoSC	500	5	0.009	0.676	0.001	0.0305
OS	BoSC	500	10	0.009	0.666	0.000	0.0312
OS	BoSC	500	20	0.009	0.644	0.000	0.0330
OS	BoSC	500	50	0.009	0.593	0.000	0.0376
OS	BoSC	500	100	0.009	0.593	0.000	0.0376
OS	BoSC	1000	5	0.009	0.677	0.001	0.0309
OS	BoSC	1000	10	0.009	0.666	0.001	0.0315
OS	BoSC	1000	20	0.009	0.660	0.000	0.0318
OS	BoSC	1000	50	0.009	0.593	0.000	0.0378
OS	BoSC	1000	100	0.009	0.593	0.000	0.0377
OS	BoSC	5000	5	0.009	0.680	0.004	0.0340
OS	BoSC	5000	10	0.009	0.670	0.002	0.0336
OS	BoSC	5000	20	0.009	0.667	0.001	0.0329
OS	BoSC	5000	50	0.009	0.601	0.001	0.0381
OS	BoSC	5000	100	0.009	0.586	0.000	0.0394
OS	STIDE	500	5	0.009	0.676	0.001	0.0308
OS	STIDE	500	10	0.009	0.666	0.001	0.0313
OS	STIDE	500	20	0.009	0.651	0.000	0.0325
OS	STIDE	500	50	0.009	0.593	0.000	0.0376
OS	STIDE	500	100	0.009	0.593	0.000	0.0376
OS	STIDE	1000	5	0.009	0.677	0.002	0.0314
OS	STIDE	1000	10	0.009	0.666	0.001	0.0318
OS	STIDE	1000	20	0.009	0.663	0.001	0.0317
OS	STIDE	1000	50	0.009	0.593	0.000	0.0378
OS	STIDE	1000	100	0.009	0.593	0.000	0.0377
OS	STIDE	5000	5	0.009	0.680	0.005	0.0354
OS	STIDE	5000	10	0.009	0.670	0.003	0.0343
OS	STIDE	5000	20	0.009	0.670	0.002	0.0332
OS	STIDE	5000	50	0.009	0.601	0.001	0.0385
OS	STIDE	5000	100	0.009	0.588	0.000	0.0393