Performance Evaluation of Container-Level Anomaly-Based Intrusion Detection Systems for Multi-Tenant Applications Using Machine Learning Algorithms

Marcos A. O. Cavalcanti*
Instituto de Telecomunicações and
Universidade da Beira Interior
Covilhã, Portugal
marcos.aurelio.oliveira.cavalcanti@ubi.pt

Pedro R. M. Inácio
Instituto de Telecomunicações and
Universidade da Beira Interior
Covilhã, Portugal
inacio@di.ubi.pt

Mário M. Freire
Instituto de Telecomunicações and
Universidade da Beira Interior
Covilhã, Portugal
mario@di.ubi.pt

ABSTRACT

The virtualization of computing resources provided by containers has gained increasing attention and has been widely used in cloud computing. This new demand for container technology has been growing and the use of Docker and Kubernetes is considerable. According to recent technology surveys, containers are now mainstream. However, currently, one of the major challenges rises from the fact that multiple containers, with different owners, may cohabit on the same host. In container-based multi-tenant environments, security issues are of major concern. In this paper we investigate the performance of container-level anomaly-based intrusion detection systems for multi-tenant applications. We investigate the use of Bag of System Calls (BoSC) technique and the sliding window with the classifier and we consider eight machine learning algorithms for classification purposes. We show that among the eight machine learning algorithms, the best classification results are obtained with Decision Tree and Random Forest which lead to an F-Measure of 99.8%, using a sliding window with a size of 30 and the BoSC algorithm in both cases. We also show that, although both Decision Tree and Random Forest algorithms leads to the best classification results, the Decision Tree algorithm has a shorter execution time and consumes less CPU and memory than the Random Forest.

KEYWORDS

Containers, Intrusion Detection Systems, System Calls, Machine Learning Algorithms

ACM Reference Format:
Marcos A. O. Cavalcanti, Pedro R. M. Inácio, and Mário M. Freire. 2021. Performance Evaluation of Container-Level Anomaly-Based Intrusion Detection Systems for Multi-Tenant Applications Using Machine Learning Algorithms. In *The 16th International Conference on Availability, Reliability and Security (ARES 2021), August 17–20, 2021, Vienna, Austria.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3465481.3470066

*Corresponding author.

ARES 2021, August 17–20, 2021, Vienna, Austria

ACM ISBN 978-1-4503-9051-4/21/08.
https://doi.org/10.1145/3465481.3470066

1 INTRODUCTION

Virtualization of computing resources can be provided through containers. This approach has practical interest since it does not depend on the base system where the applications run, allowing multiple containers with different structures and applications to run simultaneously and still share the same kernel and operating system. Container technology allows software to be deployed efficiently on a large scale. Recently, container-based virtualization has gained increasing attention, and has been widely used in cloud computing [3]. A container is a standard unit of software that packages code and all its dependencies so that the application runs quickly and reliably in a computing environment [9].

Containers have become a standard technology for applications arranged in the microservice architecture due to the light virtualization technology through the grouping and isolation of resources. Therefore, the application of intrusion detection systems in these environments is of fundamental importance, as container technology is gaining more and more strength in individual and industrial use, and, as these containers can be integrated into mission-critical systems, detecting real-time malicious cyber attacks becomes a critical operational requirement [1].

Container virtualization makes possible multiple users to use the same host, in a multi-tenant way. This is possible due to the Linux kernel service resources such as **namespaces** and **cgroups**, which are managed directly by the system kernel. Applications that run on this type of technology share virtual libraries and resources orchestrated by the operating system.

In addition to containerized applications, microservice architectures can also be used to build critical systems if the execution is supported by secure containers [10]. The microservice architecture [18] is a Software Engineering paradigm that promotes the decomposition of code-heavy monolithic applications in multiple, smaller and self-contained microservices and containers perform better in microservices-based software architectures. Unlike monolithic architectures, the separation and division of labor allows services to continue running even if the others fail, which keeps the application as a whole more reliable. However, there are security concerns due to the complexity of the security configuration of the microservices architecture and due to the fact that these resources and structures may be shared in multi-tenant environments. Containers or malicious applications that share the same environment can attack other containers, the operating system of the host or other hosts. Therefore, for this kind of applications, there is a need

for a complementary security solution to monitor these malicious activities through intrusion detection tools.

Some approaches have been published addressing the problem of intrusion detection at a container level for multi-tenancy applications [1, 12, 24], as reported in Section 2. However, these approaches present limited performance. Therefore, this paper investigates the performance of container-level intrusion detection for multi-tenant applications using several machine learning algorithms and exploring the use of the Bag of System Calls technique and a sliding window.

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 presents the experimental environment and the classification approach. Section 4 presents and discusses the results and Section 5 presents the main conclusions.

2 RELATED WORK

This section provides an overview of research works addressing intrusion detection at a container level. Abed, Clancy and Levy [1] proposed a host-based intrusion detection system for Linux containerized environments. Their intrusion detection system combines the sliding window technique with the Bag of system calls (BoSC) algorithm. This system uses system call packets monitored from the host kernel to learn the behavior of an application running on a Linux container and to evaluate anomalous container behavior. The obtained results show a 100% detection rate and a low false positive rate of 0.58%.

Flora and Antunes [12] presented a host-based intrusion detection system for containerized environments using Docker and Linux technologies. The purpose of their sustem is to collect and analyze system calls using the Sequence Time-Delay Embedding (STIDE) and Bag of System Calls (BoSC) algorithms. In a later phase, the training of the STIDE and BoSC classifiers take place, each with a window size ranging from 3 to 6 of system calls. The obtained results show a stable learning state for STIDE with window sizes between 3 and 4, and ranging from 3 to 6 for BoSC.

Srinivasan et al. [24] developed a real-time anomaly identification model that uses n-grams of system calls and the probability of their occurrence in Docker containers. The trace is processed using Maximum Likelihood Estimator (MLE) and Simple Good Turing (SGT) to provide a better estimate of system call sequence values. They used the dataset from the University of New Mexico to validate the approach with an accuracy between 87% and 97%.

Torkura et al. [25] provided a risk analysis methodology consisting of multilayered techniques that evaluate microservice architectures with resulting security risk metrics. These metrics are computed using two risk models: OWASP Risk Rating Methodology and Common Vulnerability Score System (CVSS). The techniques Moving Target Defenses (MTD) transform specified components of the system to create uncertainty for the attackers, thus reducing the likelihood of successful attacks, that is, the ability to attack. The evaluation shows that microservices are suitable for hiring Moving Target Defenses techniques with efficiency, where more than 70% of the attack surface is randomized, thus improving security.

Huang et al. [15] provided a detailed analysis of security mechanisms in the Docker container. Their approach considers two security aspects: Docker image security and container instance security. In the former, they detected vulnerabilities and exposures of pre-installed applications by the Common Vulnerabilities and Exposures (CVE) database. In the second, the instance executed the container instance, their approach monitores two aspects: use of computer resources and IP/DNS requests.

3 EXPERIMENTAL ENVIRONMENT AND CLASSIFICATION APPROACH

3.1 Overview and characteristics of the experimental environment

The experimental environment consists of a virtual machine (VM) with the Ubuntu operating system version 18.04.5 LTS (Bionic Beaver). This VM was created in a host with Windows 10 Enterprise using VirtualBox. The virtualization environment at the operating system level was implemented using the Docker container in version 20.10.03, since it is the leading container technology [5, 11].

The architecture for the container-level intrusion detection system proposed in this paper consists of the following components: container, data collection, data processing, and classifier. Figure 1 ilustrates a schematic representation of the experimental apparatus and the architecture of the container-level intrusion detection system. Each component is arranged according to its functionality and importance for the suitable operation of the intrusion detection system and for generation of a representative dataset of benign and malicious container behaviors for training and testing of the machine learning algorithms. The data processing and classifier components were implemented using the Python programming language and its libraries.

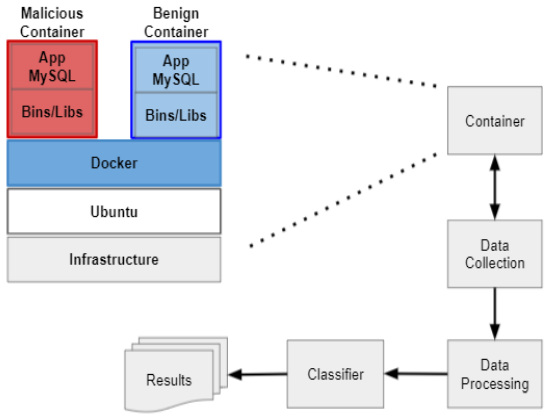

Figure 1: Overview of the experimental environment.

The characteristics of the hardware used in the experimental apparatus are provided in Table 1.

3.2 Containers

Containers provide environments for running monolithic applications or microservices, which generate benign and malicious workloads. At run time, a microservice instance can be configured to run as a process in a container [4]. The images used for generating the services are obtained from DockerHub [7], which is a

Table 1: Hardware characteristics.

Resources	Characteristics
Device	DESKTOP-MP70HAF
CPU	Intel Core i7-7700 3.6Ghz
RAM	16 GB
Hard Disk	930 GB
Host operating System	Windows 10 Enterprise
VM resources	2 cores, 4GB of RAM
VM guest operating system	Ubuntu 18.04.5 LTS

service provided by Docker. The Docker Hub is the largest official container image repository with a variety of content sources, including container community developers and open source projects. The images are from MySQL [8] which is a widely used open source relational database management system in versions 8.0.23 for benign container and 5.6.27 for malicious container.

An implementation of the TPC-C Benchmark [14, 20] was responsible for loading the container databases with 100 warehouses each and generating the online transaction services with the given number of warehouses. The TPC-C Benchmark features are defined as an online transaction processing system used to record the activities of a wholesale company. This company is distributed over a number of warehouses, districts and stock for these warehouses, as well as items and orders for that item. The number of warehouses is the key configurable parameter for determining the scale of benchmark execution. The container databases were named in the following format `tpcc100_benign` and `tpcc100_malicious`.

The simulation of the malicious behavior was designed to be composed by the search for the vulnerabilities of MySQL image, 5.6.27 version, in the CVE-Common Vulnerabilities and Exposures [6] and Exploit Database [23]. This search selected *exploits*, which are defined as a piece of software, a piece of data, or a sequence of commands that take advantage of a security flaw or vulnerability in an application or system. In this research work, we chose exploits that provide remote and local access through the following attacks: Authentication Bypass, DoS Overflow, System User Privilege Escalation/Race Condition and Integer Overflow.

3.3 Data collection

Monitoring system calls is a widely used technique for detecting suspicious behavior of an application [22]. Therefore, the data defined here for the characterization of benign and malicious container behavior are the system calls. A system call is the resource in which an application or process requests a service from the operating system kernel.

For the collection of system calls we used a system troubleshooting, analysis and exploration tool called sysdig. This diagnostic tool has an execution module for writing, reading, and storing system calls. The sysdig tool contains execution parameters through filters and chisels. In this work, the tool is run in write mode and with a specified filter parameter `container.name` followed by the name of the container to be monitored, in this case benign or malicious, and a redirect to a `csv` file.

The procedure for simulating and collecting benign behavior is obtained by running the TPC-C Benchmark implementation of operations on the `tpcc100_benign` database along with the sysdig tool for collecting the system calls. Similarly this procedure is applied for simulation of the malicious behavior by changing the `tpcc100_malicious` database and adding the vulnerabilities of MySQL image version 5.6.27.

3.4 Data processing

Data processing is a process that seeks to improve the quality of the data collected from benign and malicious container behavior. In this step, based on the knowledge-discovery in databases model, data preprocessing is performed, which includes data cleaning activities, attribute selection, as well as corrections for missing or inconsistent information so as not to compromise the quality of the models.

The `csv` files containing the system calls regarding benign and malicious behavior were reformatted to separate the obtained data in accordance with the selected attributes. According to the documentation of the sysdig [17] tool, by default, it prints the information of each captured event on a single line with the format shown in figure 2, highlighting the system calls.

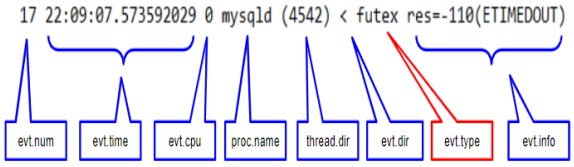

Figure 2: Format of sysdig information.

This print pattern served as a guideline for naming the attributes for the `csv` files with the system calls. For the benign behavior csv file, the columns *label* with a value of 0 (zero) and *behavior* with a value of benign were added. Similarly, for the malicious behavior `csv` file, two columns were added: a *label* column filled with value equal to 1 (one) and a *behavior* column filled with one of the values shown in Table 2 for the corresponding vulnerability.

Table 2: Classification of malicious behavior.

ID CVE	Vulnerabilities	Behavior
2012-2122	Authentication Bypass	malicious_1
2013-1861	DoS Overflow	malicious_2
2016-5616	System User Privilege Escalation	malicious_3
2017-3599	Integer Overflow	malicious_4

3.5 Classifier

A schematic representation of the architecture of the classifier used to classify the system calls as bening or malignant is illustrated in Figure 3. The classifier can operate in three different configurations: A) classification of system calls using Label Encoder and One Hot Encoder, B) classification of system calls using a sliding window

with Label Encoder and One Hot Encoder and C) classification of calls using a sliding window and the Bag of System Calls (BoSC) algorithm.

For A) and B) configurations, it is necessary to transform the categorical variables into numeric ones using the Label Encoder and One Hot Encoder methods, which are the encoders available in the Scikit-learn library [19], while for C) configuration, the variables are in numeric format.

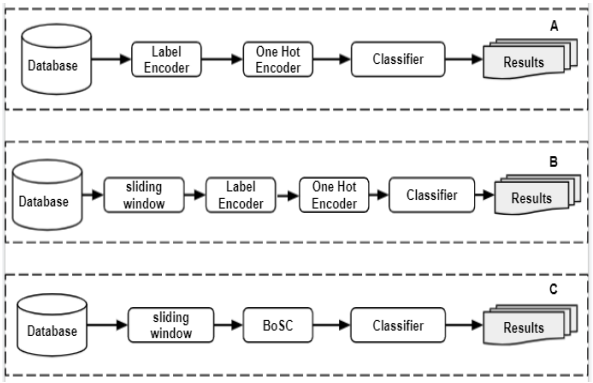

Figure 3: Classifier architecture with three possible configurations using: A) Label Encoder and One Hot Encoder, B) Sliding window with Label Encoder and One Hot Encoder, and C) Sliding window with Bag of System Call Algorithm (BoSC).

The BoSC technique is defined in Kang, Fuller and Honavar [16] as an ordered list (c1, c2, c3,..., cm) where m is the total number of distinct system calls and cj is the number of occurrences of the system call, sj , in the input sequence Zj. The sliding window technique is particular important for real-time or timely classification purposes and is based on the method proposed by Forrest et al. [13].

The classifier separately uses the following machine learning algorithms: AdaBoost (AB), Decision Tree (DT), Gaussian Naive Bayes (GNB), K-Nearest Neighbors (KNN), Multi-layer Perceptron (MLP), Multinomial Naive Bayes (MNB), Random Forest (RF), Support Vector Machines (SVM). These eight algorithms are used to classify system calls as bening or malicious. We use the implementation of these eight algorithms provided by scikit-learn [19] with the settings summarized in Table 3. The default configurations provided by scikit-learn have been used for each algorithm, with the changes in parametes optimized in Section 4.3.

4 PERFORMANCE EVALUATION

4.1 Datasets used for performance evaluation

Using the experimental apparatus described in Section 3.1 and the processes defined in Sections 3.3 and 3.4, we run experiments with the container with benign behavior during 2h05m40s and we obtained 23,774,746 calls with a size of 3.19 GB. Similarly, we run experiments with the container with malicious behavior during 2h05m40s and we obtained 18,923,927 calls with a size of 2.909 GB.

In order to reduce the execution time for the huge volume of data regarding system calls, the dataset chosen to train and test the machine learning algorithms is a subset of the stored data. Special care was taken to choose a data volume with a clear balance between samples from different classes, that is, a balanced dataset. The dataset has 200,000 instances, or data, in which 100,000 system call records of benign behavior and another 100,000 records of malicious behavior divided equally (25,000 records) across the four vulnerabilities. The dataset was divided into 60% for training and 40% for testing.

From this data set, only the attributes (resources) needed to train and test the machine learning algorithms were selected, in this case, *evt_type* and *label*. This procedure is important to minimize the processing cost and to eliminate model noise.

Table 3: Machine learning algorithms and their settings.

Algorithms	Settings
AB	(n_estimators=50, random_state=40)
DT	(criterion='gini', splitter='best', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, min_impurity_decrease=0.0, ccp_alpha=0.0)
GNB	(priors=None, var_smoothing=1e-09)
KNN	(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=None, n_neighbors=3, p=2, weights='uniform')
MLP	(alpha=1e-05, hidden_layer_sizes=(5,2), max_iter=2000, random_state=1, solver='lbfgs')
MNB	(alpha=1.0, fit_prior=True, class_prior=None)
RF	(n_estimators=100, random_state=40)
SVM	(C=1.0, kernel='linear', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=- 1, decision_function_shape='ovr', break_ties=False, random_state=None)

4.2 Performance metrics

To evaluate the performance of the classifier, we use Precision, Recall and F-Measure, which are defined in [2]. The F-Measure metric is the harmonic mean between Precision and Recall. Since this measure is an average, it gives a more accurate view of the efficiency of the classifier than merely Precision or Recall. Therefore, these metrics are used for evaluating the performance of the proposed approaches to classify the intrusion detection systems considered in this work.

4.3 Classifier optimization

In the case of the AdaBoost, K-Nearest Neighbors, Random Forest and Support Vector Machines algorithms, some specific parameters were investigated in order to optimize the classification, in terms of Precision and Recall, using the dataset mentioned in Section 4.1 without the sliding window and without the BoSC algorithm.

For the selected parameters, the range to perform the parameter optimization of these algorithms varies between 1 and 40 for Support Vector Machines and K-Nearest Neighbors algorithms, varies between 1 and 50 for the AdaBoost algorithm and from 1 to 100 for the Random Forest algorithm.

Figure 4 illustrates the evolution of the `n_estimators` parameter of the AdaBoost algorithm. According to the Scikit-learn [19] documentation, the parameter `n_estimators` is defined as the maximum number of estimators at which the reinforcement is terminated. In the case of a perfect fit, the learning process is stopped early.

As we can see in Figure 4, there is a point of intersection between the Precision and Recall for the value of `n_estimators` = 2.5. As we can see in this figure, in the range of approximately 2.5 < `n_estimators` < 5, a value for Precision above 75% and Recall above 86% can be reached. From `n_estimators` larger than 5, the Precision and Recall become constant with Precision above 70% and Recall above 85%. Thus, a value of `n_estimators` = 50 can yield values for the Recall above 85% and for Precision above 75%.

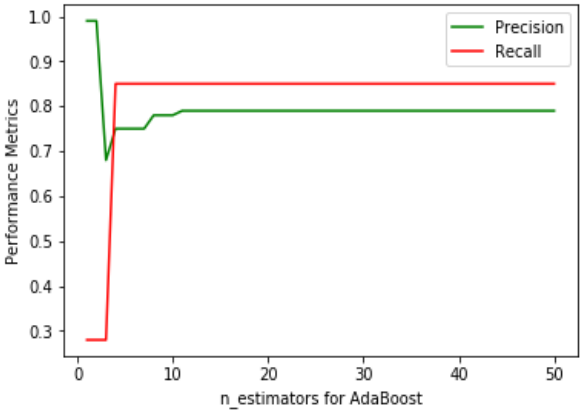

Figure 4: Evolution of the parameter `n_estimators` for AbaBoost.

Figure 5 shows the influence for Precision and Recall of the `n_neighbors` parameter of the K-Nearest Neighbors, which represents the number of neighbors required for each sample. In the range of 0<*n_neighbors*<5 the Precision and Recall metrics have different behaviors. The Precision increases in this range with values ranging from 85% to 95% and the Recall decreases in the range of 81% to 60%. In this interval there is a point of intersection between Precision and Recall.

For values of the `n_neighbors` parameter approximately larger than 2, a slight increase is observed for Recall with a value above 60%. For values of the `n_neighbors` parameter larger than 5, small variations can be observed in the values of Recall, in the range of 60% - 63%. The Precision, for values of the `n_neighbors` approximately larger than 2, decreases from 95% to 90%. From this point forward, the Precision shows small variations within this range. Therefore, with a `n_neighbors` value equal to or less than 3, values for Recall above 83% and Precision above 93% may be obtained.

Figure 6 shows the performance of Precision and Recall as a function of the `n_estimators` parameter that is used to control the

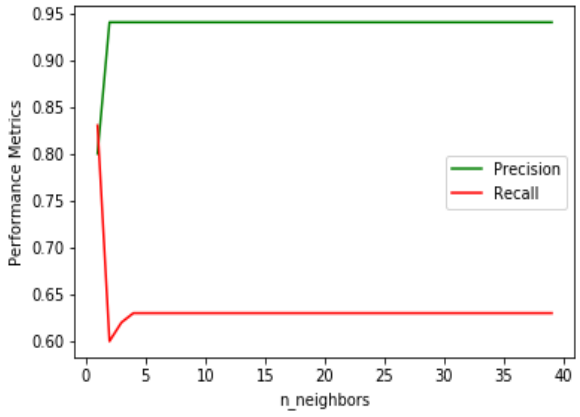

Figure 5: Evolution of `n_neighbors` for K-Nearest Neighbors.

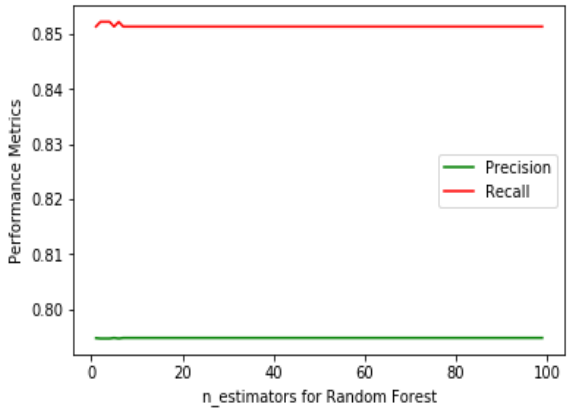

Figure 6: Evolution of the `n_estimators` parameter for Random Forest.

number of trees in the process for the Random Forest algorithm. For 0<`n_estimators`<5, the Recall shows an increasing behavior with values between 85% and 86% and the Precision shows a constant behavior with a value below 80%. Therefore, we consider a value of `n_estimators` = 100 to keep the values of Recall above 90% and Precision above 80%.

Figure 7 shows the values for Recall and Precision as a function of the `C` parameter, which is the regulation parameter of the Support Vector Machines algorithm. For values of `C` parameter larger than 3, the Precision is almost constant with a value of around 79% and the Recall is almost constant with a value around 85%.

4.4 Results for Label Encoder and One Hot Encoder

The results shown in table 4 were obtained by the application of Label Encoder and One Hot Encoder provided by Scikit-learn, which are used to convert categorical data (system calls) into numerical data.

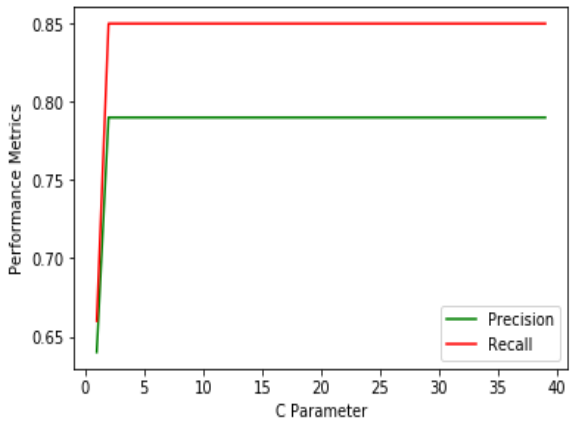

Figure 7: Evolution of C-Parameter for SVM.

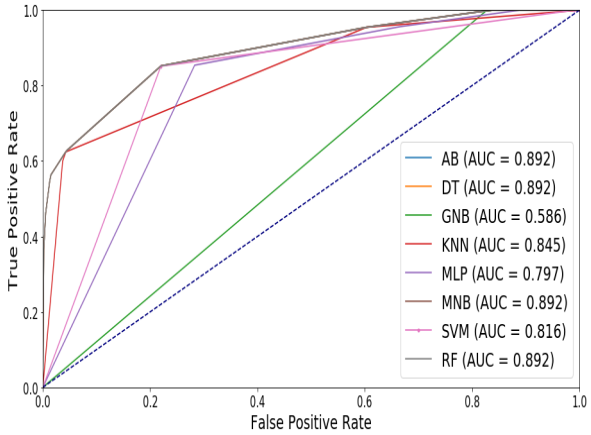

Figure 8: ROC curve for machine learning algorithms with Label Encoder and One Hot Encoder.

Table 4: Performance evaluation of machine learning algorithms with Label Encoder and One Hot Encoder.

	Metrics		
Classifier	Precision	Recall	F-Measure
AdaBoost	0.794	0.852	0.822
Decision Tree	0.794	0.851	0.822
Gaussian Naive Bayes	0.547	0.997	0.707
K-Nearest Neighbors	0.936	0.624	0.749
Multi-layer Perceptron	0.751	0.853	0.799
Multinomial Naive Bayes	0.794	0.851	0.822
Random Forest	0.794	0.851	0.822
Support Vector Machine	0.795	0.855	0.832

As we can see in this table, Precision ranges from 54.7% to 93.6%, being the highest value obtained with the K-Nearest Neighbors algorithm, while Recall varies between 62.4% and 99.7%, being the highest value obtaned with Gaussian Naive Bayes. Regarding F-Measure, it ranges from 70.7% to 83.2%, being the highest value obtained for Support Vector Machine. Figure 8 shows the corresponding ROC (Receiver Operating Characteristic) curves, for comparison purposes. As we can see in this figure, AbaBoost, Decision Tree, Multinomial Naive Bayes and Random Forest algorithms lead to an AUC (Area Under Curve) of 0.892. These four algorithms lead to the same F-Measure of 82.2%, slightly below of the value of 83,2% obtained for Support Vector Machine.

4.5 Results for sliding window technique

Table 5 presents the classification results for the eight machine learning algorithms using the sliding window, but without using the BoSC technique. The size of the window ranges from 5 to 30 in multiples of 5. As we can see in this table, the best result for F-Measure is obtained for Random Forest with 99.4%, followed by Support Vector Machines with 99.3%, with a window size of 30 in both cases. Figure 9 shows the ROC curves for operation with a sliding window with size of 30. The Random Forest algorithm presents again the best result with AUC = 1.0. The Multi-layer Perceptron and Support Vector Machine algorithms have both AUC=0.999 and the Multinomial Naive Bayes has an AUC=0.998.

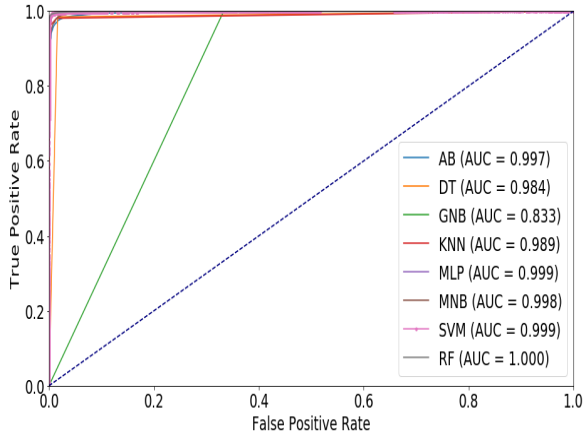

Figure 9: ROC curve for machine learning algorithms with a sliding window with size of 30.

4.6 Results for sliding window and BoSC techniques

Table 6 presents the classification results for the eight machine learning algorithms using a sliding window jointly with the BoSC technique. The size of the sliding window ranges from 5 to 50 in multiples of 5.

Table 5: Performance evaluation of machine learning algorithms with sliding window technique.

Classifier	Metrics	Size of the Sliding Window					
		5	10	15	20	25	30
AdaBoost	Precision	0.968	0.973	0.977	0.981	0.979	0.981
	Recall	0.902	0.940	0.960	0.964	0.973	0.976
	F-Measure	0.934	0.956	0.968	0.972	0.976	0.979
Decision Tree	Precision	0.972	0.975	0.978	0.982	0.983	0.985
	Recall	0.934	0.968	0.976	0.980	0.982	0.985
	F-Measure	0.952	0.972	0.977	0.981	0.983	0.985
Guassian Naive Bayes	Precision	0.613	0.654	0.682	0.709	0.732	0.751
	Recall	0.998	1.000	1.000	1.000	1.000	1.000
	F-Measure	0.759	0.791	0.811	0.830	0.845	0.858
K-Nearest Neighbors	Precision	0.958	0.978	0.985	0.992	0.994	0.996
	Recall	0.939	0.959	0.966	0.966	0.966	0.964
	F-Measure	0.958	0.969	0.976	0.978	0.980	0.980
Multi-layer Perceptron	Precision	0.968	0.981	0.988	0.992	0.993	0.993
	Recall	0.931	0.967	0.980	0.986	0.989	0.990
	F-Measure	0.949	0.974	0.984	0.989	0.991	0.991
Multinomial Naive Bayes	Precision	0.963	0.975	0.980	0.987	0.988	0.989
	Recall	0.896	0.937	0.951	0.962	0.968	0.971
	F-Measure	0.928	0.956	0.965	0.974	0.978	0.980
Random Forest	Precision	0.971	0.981	0.988	0.993	0.994	0.996
	Recall	0.937	0.975	0.984	0.986	0.989	0.991
	F-Measure	0.954	0.978	0.986	0.989	0.992	0.994
Support Vector Machine	Precision	0.960	0.976	0.987	0.990	0.992	0.994
	Recall	0.923	0.961	0.978	0.984	0.989	0.992
	F-Measure	0.941	0.969	0.982	0.987	0.990	0.993

As we can see in this table, the best results of F-Measure are obtained for Decision Tree and Random Forest, both with 99.8%, using a sliding window with a size 30. In the same conditions, K-Nearest Neighbors leads to an F-Measure of 99.7%. Figure 10 shows the ROC curves for operation with a sliding window with size of 30 and BoSC algorithm. In this case, AdaBoost, Multi-layer Perceptron, Support Vector Machine and Random Forest classifiers show the best result with AUC=1.0, but Multi-layer Perceptron has an F-Measure of 99.6% and AdaBoost and Support Vector Machine classifiers have an F-Measure of 99.4%.

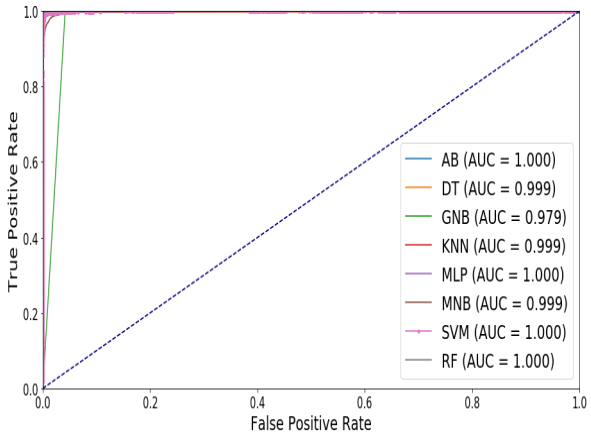

Figure 10: ROC curve for machine learning algorithms with a sliding window with size of 30 and BoSC.

4.7 Resource usage

The table 7 shows the resources used for each classifier with respect to CPU, RAM, and execution time in the training and testing processes using the datasets considered in this paper. The command GNU/Linux/`usr/bin/time` provides the CPU consumption and execution times. The RAM usage was collected using the psrecord software [21]. In this table we consider sliding windows with sizes of 30 because this window size leads to the best classification results among the window size span considered in this paper.

Table 6: Performance evaluation of machine learning algorithms with sliding window and BoSC techniques.

Classifier	Metrics	Size of the Sliding Window					
		5	10	15	20	25	30
AdaBoost	Precision	0.966	0.976	0.989	0.990	0.992	0.996
	Recall	0.921	0.968	0.976	0.985	0.991	0.991
	F-Measure	0.943	0.972	0.982	0.987	0.992	0.994
Decision Tree	Precision	0.977	0.988	0.992	0.996	0.997	0.999
	Recall	0.920	0.966	0.984	0.990	0.994	0.996
	F-Measure	0.948	0.977	0.988	0.993	0.996	0.998
Guassian Naive Bayes	Precision	0.606	0.654	0.689	0.714	0.732	0.751
	Recall	0.999	0.998	0.999	1.000	1.000	1.000
	F-Measure	0.754	0.790	0.816	0.833	0.845	0.858
K-Nearest Neighbors	Precision	0.964	0.984	0.990	0.995	0.996	0.998
	Recall	0.922	0.964	0.982	0.989	0.993	0.997
	F-Measure	0.943	0.974	0.986	0.992	0.995	0.997
Multi-layer Perceptron	Precision	0.969	0.980	0.990	0.992	0.993	0.997
	Recall	0.924	0.965	0.978	0.984	0.990	0.994
	F-Measure	0.946	0.972	0.984	0.988	0.992	0.996
Multinomial Naive Bayes	Precision	0.963	0.976	0.981	0.987	0.989	0.989
	Recall	0.897	0.938	0.952	0.962	0.968	0.971
	F-Measure	0.929	0.956	0.966	0.975	0.978	0.980
Random Forest	Precision	0.977	0.988	0.993	0.996	0.998	0.999
	Recall	0.921	0.967	0.985	0.991	0.995	0.997
	F-Measure	0.948	0.977	0.989	0.993	0.996	0.998
Support Vector Machine	Precision	0.968	0.974	0.988	0.991	0.993	0.995
	Recall	0.913	0.965	0.978	0.983	0.989	0.993
	F-Measure	0.940	0.969	0.983	0.987	0.991	0.994

As we can see in table 7, the shortest execution time of 7s is obtained for MLP with Label Encoder and One Hot Encoder encoders with CPU and RAM consumptions of 86% and 745 MB, respectively. For classification using a sliding window With a size of 30, the shortest execution time is obtained for Multinomial Naive Bayes with 55 s with CPU and RAM consumptions of 77% and 1.505 GB, respectively, while, for classification using a sliding window With a size of 30 and the BoSC algorithm, the shortest execution time is obtained for Decision Tree with 39 s with CPU and RAM consumptions of 89% and 913 MB, respectively.

The execution times for the best classification results, obtained for an F-Measure of 99.8% with a sliding window with size of 30 and the BoSC algorithm, are 39 s for Decision Tree, with CPU and RAM consumptions of 89% and 913 MB, and 135 s for Random Forest, with CPU and RAM consumptions of 98% and 1.163 GB. Although both algorithms lead to the same F-Measure, the Decision Tree algorithm is faster and consumes less CPU and memory.

5 CONCLUSIONS

In this paper we investigated the performance of intrusion detection systems at a container level for multi-tenant applications. We took into account the influence of the use of Bag of System Calls (BoSC) algorithm and the sliding window technique and we considered eight machine learning algorithms for classification of system calls as benign or malignant. The classification results show that Decision Tree and Random Forest algorithms lead to the highest values of the F-Measure of 99.8%, being these results achieved with a sliding window with size 30 and the BoSC algorithm. We have also investigated the resource usage by the eight machine learning algorithms in the different classifier configurations, being shown that, for the two algorithms that lead to an F-Measure of 99.8%, the Decision Tree algorithm is faster and consumes less CPU and memory than the Random Forest algorithm. For future work, we plan to optimize the size of the sliding window for the classifier performance and we plan to include more vulnerabilities and services in the multi-tenant container environment.

Table 7: Computational resource usage by algorithms.

	Label Encoder One Hot Encoder			Sliding Window n = 30			Sliding Window n = 30 and BoSC		
Classifier	CPU (%)	RAM (MB)	Time (s)	CPU (%)	RAM (MB)	Time (s)	CPU (%)	RAM (MB)	Time (s)
AdaBoost	89	873	9.5	93	1,929	120	87	794	48
Decision Tree	79	985	11.3	98	1,863	480	89	913	39
Gaussian Naive Bayes	73	1,083	8.1	79	1,689	58	85	893	44
K-Nearest Neighbors	83	596	526	93	1,754	2,820	91	1,147	1,550
Multi-layer Perceptron	86	745	7	96	1,756	384	92	1,094	244
Multinomial Naive Bayes	92	957	10	77	1,505	55	82	674	45
Random Forest	97	953	18	98	1,609	59	98	1,163	135
Support Vector Machine	89	754	1,200	98	879	2,845	98	893	300

ACKNOWLEDGMENTS

This work is funded by Portuguese FCT/MCTES through national funds and, when applicable, co-funded by EU funds under the project UIDB/50008/2020 and by FCT/COMPETE/FEDER under the project SECURIoTESIGN with reference number POCI-01-0145-FEDER-030657, and funded by operation Centro-01-0145-FEDER-000019 - C4 - Centro de Competências em Cloud Computing, co-funded by the European Regional Development Fund (ERDF/FEDER) through the Programa Operacional Regional do Centro (Centro 2020).

REFERENCES

[1] Amr S. Abed, Charles Clancy, and David S. Levy. 2015. Intrusion Detection System for Applications Using Linux Containers. *Lecture Notes in Computer Science* (2015), 123–135. https://doi.org/10.1007/978-3-319-24858-5_8

[2] N. Antunes and M. Vieira. 2015. On the Metrics for Benchmarking Vulnerability Detection Tools. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 505–516. https://doi.org/10.1109/DSN.2015.30

[3] L. Cai, Y. Qi, W. Wei, and J. Li. 2019. Improving Resource Usages of Containers Through Auto-Tuning Container Resource Parameters. *IEEE Access* 7 (2019), 108530–108541. https://doi.org/10.1109/ACCESS.2019.2927279

[4] Ramaswamy Chandramouli. 2019. Security Strategies for Microservices-based Application Systems. In *NIST SP 800-204*. National Institute of Standards and Technology (NIST). U.S. Department of Commerce. https://csrc.nist.gov/publications/detail/sp/800-204/final

[5] T. Combe, A. Martin, and R. Di Pietro. 2016. To Docker or Not to Docker: A Security Perspective. *IEEE Cloud Computing* 3, 5 (2016), 54–62. https://doi.org/10.1109/MCC.2016.100

[6] Mitre Corporation. 1999. *Common Vulnerabilities and Exposures*. Retrieved January 6, 2021 from https://cve.mitre.org/

[7] Docker. 2013. *DockerHub*. Retrieved January 6, 2021 from https://www.docker.com/products/docker-hub

[8] Docker. 2013. *mysql-Docker Official Images*. Retrieved January 6, 2021 from https://hub.docker.com/_/mysql

[9] Docker. 2021. *What is a container? a standardized unit of software*. Retrieved March 2, 2021 from https://www.docker.com/resources/

[10] C. Fetzer. 2016. Building Critical Applications Using Microservices. *IEEE Security & Privacy* 14, 06 (nov 2016), 86–89. https://doi.org/10.1109/MSP.2016.129

[11] Flexera. 2021. *Flexera 2021 State of the Cloud Report*. Retrieved January 6, 2021 from https://info.flexera.com/CM-REPORT-State-of-the-Cloud?lead_source=Website%20Visitor&id=Flexera.com-PR

[12] J. Flora and N. Antunes. 2019. Studying the Applicability of Intrusion Detection to Multi-Tenant Container Environments. In *2019 15th European Dependable Computing Conference (EDCC)*. 133–136. https://doi.org/10.1109/EDCC.2019.00033

[13] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. 1996. A sense of self for Unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*. 120–128. https://doi.org/10.1109/SECPRI.1996.502675

[14] Amitabh Shah Francois Raab, Walt Kohler. 1992. *Overview of the TPC-C Benchmark. The Order-Entry Benchmark*. Retrieved March 2, 2021 from http://www.tpc.org/tpcc/detail5.asp

[15] D. Huang, H. Cui, S. Wen, and C. Huang. 2019. Security Analysis and Threats Detection Techniques on Docker Container. In *2019 IEEE 5th International Conference on Computer and Communications (ICCC)*. 1214–1220. https://doi.org/10.1109/ICCC47050.2019.9064441

[16] Dae-Ki Kang, D. Fuller, and V. Honavar. 2005. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. 118–125. https://doi.org/10.1109/IAW.2005.1495942

[17] Michael Kerrisk. 2021. *sysdig(8) — Linux manual page*. Retrieved March 2, 2021 from https://man7.org/linux/man-pages/man8/sysdig.8.html

[18] Sam Newman. 2015. *Building Microservices* (1st ed.). O'Reilly Media, Inc.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[20] Percona-Lab_tpcc-mysql. 2017. . Retrieved January 6, 2021 from https://github.com/Percona-Lab/tpcc-mysql

[21] psrecord. 2021. *Record the CPU and memory activity of a process*. Retrieved April 6, 2021 from https://github.com/astrofrog/psrecord

[22] M. Rajagopalan, M. A. Hiltunen, T. Jim, and R. D. Schlichting. 2006. System Call Monitoring Using Authenticated System Calls. *IEEE Transactions on Dependable and Secure Computing* 3, 3 (2006), 216–229. https://doi.org/10.1109/TDSC.2006.41

[23] Offensive Security. 2010. *Exploit Database*. Retrieved January 6, 2021 from https://www.exploit-db.com/

[24] Siddharth Srinivasan, Akshay Kumar, Manik Mahajan, Dinkar Sitaram, and Sanchika Gupta. 2019. *Probabilistic Real-Time Intrusion Detection System for Docker Containers: 6th International Symposium, SSCC 2018, Bangalore, India, September 19–22, 2018, Revised Selected Papers*. 336–347. https://doi.org/10.1007/978-981-13-5826-5_26

[25] K. A. Torkura, M. I. H. Sukmana, A. V. D. M. Kayem, F. Cheng, and C. Meinel. 2018. A Cyber Risk Based Moving Target Defense Mechanism for Microservice Architectures. In *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. 932–939. https://doi.org/10.1109/BDCloud.2018.00137