Evaluation of Machine Learning for Intrusion Detection in Microservice Applications

Iury Araujo
University of Coimbra
CISUC, DEI
Coimbra, Portugal
iuryrogerio@dei.uc.pt

Nuno Antunes
University of Coimbra
CISUC, DEI
Coimbra, Portugal
nmsa@dei.uc.pt

Marco Vieira
University of Coimbra
CISUC, DEI
Coimbra, Portugal
mvieira@dei.uc.pt

ABSTRACT

Microservices have thrived recently as an approach for service design, development, and delivery. It provides several benefits to the systems as an architecture, such as faster delivery, improved scalability, and greater autonomy. Although microservice architectures are popular, security characteristics of these architectures impair the deployment of security, such as sizable attack surface, network complexity, heterogeneity, and others. For years, intrusion detection has been a practical security approach for many applications. Recently, machine learning provided improved functionality for intrusion detection systems with exciting results in overall tests. This paper presents the evaluation of machine learning techniques for intrusion detection in a microservice scenario. System call data was collected from containers simulating microservice applications; these containers were submitted to attacks that exploited different vulnerabilities. The data was used to train and test machine learning techniques, and the test results provided us with exciting possibilities for this approach. Some of the tested attacks were very well detected by the techniques, while some were not, attesting that machine-learning-based intrusion detection is usable in this environment. However, to enhance detection, it is required to improve data processing and representation for this type of scenario.

CCS CONCEPTS

• **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**; • **Computing methodologies** → *Machine learning approaches.*

KEYWORDS

Intrusion Detection, Machine Learning, Microservices, System Calls

ACM Reference Format:
Iury Araujo, Nuno Antunes, and Marco Vieira. 2023. Evaluation of Machine Learning for Intrusion Detection in Microservice Applications. In *12th Latin-American Symposium on Dependable and Secure Computing (LADC 2023), October 16–18, 2023, La Paz, Bolivia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3615366.3615375

LADC 2023, October 16–18, 2023, La Paz, Bolivia

ACM ISBN 979-8-4007-0844-2/23/10.
https://doi.org/10.1145/3615366.3615375

1 INTRODUCTION

Microservices have become a popular approach in recent years [12, 15, 26]. They provide an approach to software and systems architecture that builds on the firmly established concept of modularization [12]. Microservices are small and independent applications responsible for a single goal and can be deployed, scaled, and tested independently [15]. Thus, the approach renders monolithic systems into granular units that interact via messages through a well-defined network interface, increasing software agility and enabling organizations to make more continuous deliveries and achieve better time to market [12, 15].

Although microservices provide several advantages to systems development, new challenges appear, such as security [19]. For instance, recent industrial reports reveal that big companies have been receiving massive attacks on their microservices-based systems and infrastructure in the last few years [19]. One such company is Netflix which had a subdomain compromised, and the attacker was able to send any content via the context netflix.com to the users [6, 19]. The security issues microservices face are not strictly new, being present in Service-Oriented Architecture and generally in distributed systems. However, they become more challenging due to specific security characteristics of the microservices environment, such as greater attack surface, network complexity, and heterogeneity, thus remaining open challenges [6, 25].

Several studies have proposed solutions for the issues discussed earlier [8, 11, 17]. However, as technologies evolve, attacks and vulnerabilities do so as well. Therefore, security approaches must be revised, improved, and sometimes created anew. Also, emergent technologies can be used to propose new approaches. One such technology is Machine Learning, which has become popular in improving the security of distributed architectures, such as edge computing and IoT [4, 7, 20, 24]. In these architectures, ML is used as the backbone of the attack identification process from Intrusion Detection System, exploring the data of the target applications [5]. Although ML techniques can help to improve security in distributed architectures, not all methods provide good results. Many of them need help from data processing before using the ML methods or will need post-processing techniques to attain a satisfactory solution, and some can never work with that type of scenario. Thus, applying ML to security solutions, such as intrusion detection, is a constant study, adaptation, and testing process.

In this work, our goal is to analyze how machine learning techniques may be used to detect intrusions in applications deployed in a microservice scenario. For this, we have evaluated different ML techniques and configurations using an experiment to identify how

well these techniques could identify distinct attacks. The experiment described here used system call data collected from containers deployed in Docker to simulate standalone services. This training and testing data was submitted to pre-processing methods to transform it into a suitable technique format. Also, a post-processing technique based on a sliding window was used to improve the identification of attacks and minimize the number of false alarms. The results have shown promising possibilities for using ML techniques for intrusion detection in this scenario. However, it seems dependent on the quality of the training and testing data, including the ability to find a proper representation of system call data in this scenario.

The paper is structured as follows. Section II presents the background and the works related to the research. The evaluation process is described in section III. Section IV discusses the experiment, its configuration and the steps to obtain the training and testing data. The results will be presented and discussed in Section V. Lastly, Section VII concludes the work and presents the future work.

2 BACKGROUND AND RELATED WORK

Intrusion detection monitors events occurring in computer systems or networks, analysing them for traces of security problems [1]. Intrusion Detection System detects malicious activities and unauthorized usage to protect computers and network infrastructure. An Intrusion Detection System will capture and analyse network traffic to discover suspicious activities and report them to an administrator, human or computational system, which decides the action to that activity [5]. Intrusion Detection System must constantly analyse activities performed on the system. These activities are the data used to identify attacks from normal behaviour. Mainly, IDS will use two approaches to analyse data, signature and anomaly detection.

Signature detection, also called misuse detection, identifies attacks by searching for a specific pattern or behaviour already classified as an attack stored in a signature database [1, 14]. This approach often uses pattern-matching techniques to identify malicious activity. Anomaly detection will search for deviations from the expected behaviour, indicating an probable attack on the system. This type of detection assumes that anomalies are rare events that are distinctive from the system's normal behaviour. Thus, it is possible to construct a model of the system's normal behaviour, and everything that deviates from that is considered an anomaly [1, 14]. The development of anomaly detection models is executed through two phases: the training phase and the testing phase.

Machine learning has become an exciting technology for intrusion detection approaches. Computer security intrinsically provides robust datasets on which ML techniques may take advantage. In addition, the advent of powerful data crushing hardware and more capable data analysis and ML algorithms provide the perfect time to explore the potential of ML in the area of security [5]. Several studies, like the ones in [2, 10, 20, 22], have provided information about the ML techniques that could be used to provide intrusion detection for a variety of scenarios and vulnerabilities. Unfortunately, it is impossible to recommend a specific ML technique based only on the system exposition to attacks. The richness and complexity of the techniques do not allow for such simple categorisation [2]. Also, studies warn of the importance of training and testing data in intrusion detection. ML techniques can not work correctly without representative data, and obtaining this data is a time-consuming and challenging task [2, 10].

Unfortunately, studies on machine learning for security in microservices are scarce. The work developed in [18] proposes a machine-learning-based approach capable of modelling microservices behaviour by observing inter-service communication. This modelling creates a visualisation of the system's expected behaviour and its services. Therefore any action that would deviate from this model is considered an anomaly and possible intrusion. Although the idea expands to microservice applications, the approach is designed to train and test the Internet of Things (IoT) services.

In [13], an ML-based detection system is proposed to analyse system call data from microservices in cloud containers to identify cryptomining action. The tests implement a real-world scenario in Kubernetes with a detection system composed of different ML techniques to identify any anomalous pods. The results show that three techniques of the four tested achieved a collective accuracy of 97%. Also the study provides exciting ideas for the organisation and collection of training and testing system call data. The work presented in [3] presented a performance evaluation of an intrusion detection system based on ML techniques for multi-tenant applications. The experiment uses different forms of embedding the system call data and shows promising results for the majority of the ML techniques tested. However, the precision, recall, and F-measure metrics were not separated for the type of attack. Thus, it is impossible to determine if the result is valid for all types or if it is just one type with good results concealing the bad results from the others.

The related works show a research gap in machine-learning-based intrusion detection systems for microservice security. Although ML techniques are being incorporated into intrusion detection systems, many researchers are focused only on IoT or edge scenarios.On the other hand, the few microservices studies we encountered usually focus on one class of vulnerability. Thus, part of our proposal focuses on providing an approach for microservice security using machine-learning-based intrusion detection for several classes of vulnerabilities.

3 EVALUATION OF MACHINE LEARNING TECHNIQUES

To assess the ML techniques, we designed a procedure where the techniques would be trained and tested, and changes could be applied based on the results. As different parameters of the algorithms may change the results of ML techniques, evaluating the results from changes in parameters is also necessary. Figure 1 presents the procedure for evaluating techniques, which is divided into phases, as follows.

As we can observe, in the first phase, a technique is selected from the available ones and paired with a set of parameters, this pair will use a percentage of the data from the scenario to train a model based on the techniques algorithm and the given parameters. The second phase will take the model created by the previous phase and a smaller percentage of the data to test the model and determine the confusion matrix for the test. In the third phase, the results

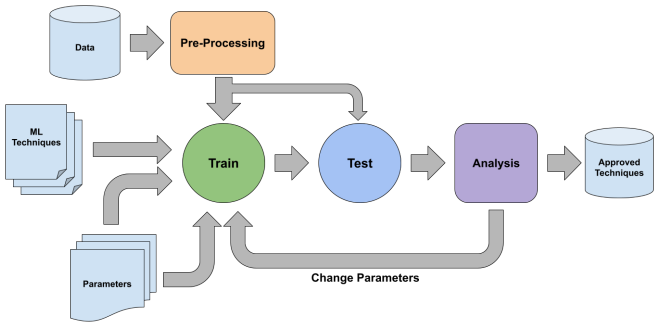

Figure 1: Evaluation procedure of machine learning techniques.

from the confusion matrix are analysed and if they are considered satisfactory, the technique is approved for that set of parameters. Also, all the information from the evaluation is saved for later use, such as parameters, training and testing data, and the confusion matrix from the test results. Lastly, the fourth phase only happens if the results are not acceptable. In this case, the technique is sent again to the first phase, and a new set of parameters is chosen to be executed with it, restarting the evaluation procedure.

The pre-processing task will mainly transform the data from microservice applications into a format and structure more applicable to the ML technique at the moment. This task may have a different number of steps depending on the technique. For example, techniques based on distance will have different pre-processing approaches than the ones based on regression. Also, the data may be refined or processed to improve the quality of the data and offer a subset more suitable for the problem. This happens in situations in which the volume of data is greater than needed, and some pre-processing techniques may be used to reduce the size of subsets. Moreover, pre-processing is also needed when the data present imbalanced characteristics in which a class, or classes, has just a tiny number of instances in the data but are very important for the context of the model. In this context, the data produced by the pre-processing techniques may help to avoid model fitting.

For this work, we chose to work with only the following supervised techniques: Decision Tree, Random Forest, and Support Vector Machine. DT and RF are techniques widely used for machine-learning-based intrusion detection in other scenarios and are relatively simple to use [2, 4, 10, 13, 24]. Thus, it seemed natural to start the processes with them. SVM is another technique frequently used in the context of intrusion detection [21, 23]. The appeal for using this technique as one of the firsts to be assessed comes from the different kernel functions that can be used by the training algorithm, which prompts different results and a deeper analysis of the impact of technique parameters on the evaluation procedure.

4 EXPERIMENTAL EVALUATION

To test the evaluation approach for ML techniques explained in Figure 1, we decided to evaluate the three techniques established previously. Each technique was trained, tested, and evaluated more than once with different parameters and changes to the pre-processing approaches. The experiment was implemented in Python 3.8.10, including pre-processing and post-processing tasks. The free library Scikit-learn version 0.24.2 was used for ML training and testing. All the phases and tasks of the evaluation procedure were conducted in a machine with the following specifications: Intel Core i5 10400 CPU at 2.90GHz with 32GB of RAM, with Ubuntu 20.04.2 LTS.

4.1 Data Collection

The quality of the data used for training ML techniques is directly related to the model's capacity to learn the scenario correctly. Thus, the data must represent the nuances of the scenario completely. We are using system call data to train and test the techniques for this work. Microservice applications can provide a variety of data usable in intrusion detection approaches. Among this variety, system call data has been widely used by Intrusion Detection System for a long time to evaluate any deviation from the normal execution, thus identifying possible security problems [16].

The data used to train and test the ML techniques in our evaluation procedure come from the work done by Flora J., Gonçalves P., and Antunes N. in [9] to evaluate intrusion detection effectiveness in cloud container-based systems. Their work used a MariaDB deployment for Docker container to simulate a microservice. Based on the TPC-C benchmark, a variable workload was used to perform requests to the container. This workload had a variable number of active clients. It starts with ten active connections but repeatedly increases until it reaches a maximum of 90 connections and then decreases until it arrives again at ten active connections. In addition, this workload would also periodically execute commands that emulate administrative tasks. The *sysdig* was used to collect the system calls because of its native support for containers.

The data was collected in the form of collections, and each collection contains system calls executed by the container during a period ($30min$). In the middle of the collection time ($15min$), an attack against the MariaDB is performed. The time interval for the attack is important because the collection can contain the system's execution before and after the attack. Five types of vulnerabilities were exploited to produce attack data for the training and testing of the ML techniques. They were vulnerabilities in an older version of MariaDB with proof-of-concept code available at exploitdb.com. For each type of vulnerability, three collections were produced. Table 1 offers an overview of the selected vulnerabilities and their identification through this work. Additionally, three collections were created with only benign information gathered over a more extensive period ($24H$).

Table 1: List of vulnerabilities exploited in this work. [9]

CVE ID	Acess	Vulnerability Type(s)	CVSSS	ID
CVE-2016-6662	Remote	Execute Code, Bypass	10.0	V1
CVE-2012-5611	Remote	Execute Code, Overflow	6.5	V2
CVE-2013-1861	Remote	Denial Of Service, Overflow	5.0	V3
CVE-2012-5627	Remote	Execute Code	4.0	V4
CVE-2016-6663	Local	Gain privileges	4.4	V5

The deployment for the collection of data in the work [9] was configured with Ubuntu 14.04 LTS and installed MariaDB version 5.5.28 with default configurations. Two machines were used, a Target Infrastructure and a Test Driver. The target infrastructure machine supported the evaluated service and has a 6-core 2.8GHz CPU with 32GB of RAM, with Ubuntu 18.04.2 LTS. The Test Driver has a 3.20GHz CPU and 16GB of RAM, with Ubuntu 16.04.3 LTS.

4.2 Metrics and Evaluation

Metrics are essential to evaluate the efficiency of the models created by the training phase and determine if the ML techniques can detect intrusions in microservice applications. They also help determine if any change to the parameters improves or worsens the model's efficiency. ML is a study field that possesses many techniques, such as accuracy, recall, precision, F1-score, AUC, ROC, and others, and their use will depend on which aspects are more important to be analysed by the research. This work has decided to use only two specific metrics: sensitivity and false alarm ratio, respectively, the True Positive Rate and False Positive Rate. These metrics can be obtained by evaluating the confusion matrix of the test phase. The decision to use these metrics was based on the necessity to evaluate the TPR during the attack period and the FPR before and after the attack.

To better evaluate the results, we divided the testing data into three periods: pre-attack, attack, and post-attack. Figure 2 presents the timeline of the testing data. What we expect from a successful model is a high TPR during the attack and a lower number of false alarms FPR during the pre-attack and the post-attack. Although this evaluation is very simplified, it answers the questions about the efficiency of the techniques and the following need to execute changes, be these changes to the parameters or the data. Also, it is possible that even with changes, some vulnerabilities are not detected by some techniques, which can be a problem related to the similarity of normal and attack instances or the quantity of data needed to produce a workable model.

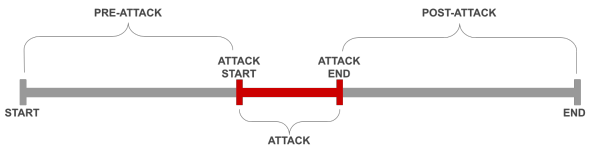

Figure 2: Division of the testing data into three time periods.

4.3 Data Pre-processing

Although the collected data is enough to represent our scenario, its format and structure are not adequate for the machine learning techniques. Thus, based on the work of [13], which uses system calls to detect cryptomining in cloud containers, we have decided to use their approach to structure system calls. The data collected are composed of the system calls and their parameters. For this experiment, we have chosen to ignore the parameters of the system calls and see no distinction among the same command with different parameters. We believe that these parameters have interesting information and may help create more accurate models in the future. However, encoding these parameters in a structure that could represent the differences among system calls is not an easy task and would require more resources.

Figure 3 explains transforming system call data into training and testing data for our study of ML techniques. The process is divided into three phases: encoding, n-gram, and feature vector. In the encoding phase, a dictionary containing all system calls from the raw data is created, and each system call has associated with a unique identification number. Next, the raw data pass through an encoding process where the system call data is transformed into a sequence of identification numbers based on the created dictionary. This encoded data maintains the same sequence of system calls present in the raw data. In the n-grams phase, the process uses the n-grams approach from [13]. This approach divides the sequence of system calls into units of size n called frames.

The value of n is entirely dependable on tests and experimentation. Initially, our strategy was based on the research in [13], which uses a frame size of $n = 35$. Although the results were at first good, we decided to evaluate the TPR during the attack when testing with other frame sizes of $n = \{35, 50, 100, 200, 400\}$. To evaluate the classification results of different frame sizes under various attacks, we have decided to use the size $n = 50$ for this experiment. This choice was based on how stable was the classification of attacks for this frame size, around 80-87% of correct attack classification. Frames have a better granularity to be evaluated by the ML techniques than using each system call because they are easier to extract features.

Lastly, the feature vector phase transforms the frames into feature vectors. This transformation helps structure the data into a more suitable format for the ML techniques. However, we no longer consider the system calls sequence in our model training by using feature vectors. Each position in the feature vector is connected to a type of system call in the dictionary. In these positions, the number of times a system call appears in the frame is recorded, and the spaces related to system calls that are not present in the frame are marked with zeros. Also, feature vectors formed by this phase will have a size equal to the number of elements in the dictionary, which permits the test of different frame sizes without changing the format.

4.4 Data Post-processing

The post-processing approach uses results from the test phase of the evaluation procedure. These results are composed of a sequence of frames classified with a binary value $f = \{0 - Normal, 1 - Attack\}$. They are organised following their timestamps, maintaining the proper timeline of frames precisely as they were collected. To process this timeline, we create a window of size n and set an attack threshold of percentage p. Thus, the window will analyse n frames per iteration and verify how many are classified as attacks if that value is equal to or larger than $x = ((n * p) \div 100)$, and an attack alert is invoked. For instance, considering a window of size $n = 50$ and a threshold of $p = 10\%$, each iteration will evaluate 50 frames. If at least five are classified as attacks, the post-processing approach will invoke an attack alert at that moment.

Besides calculating the attack alert, the approach also designs the window as a sliding window. This window will begin the timeline analysis with the first n frames. After the evaluation, it will move

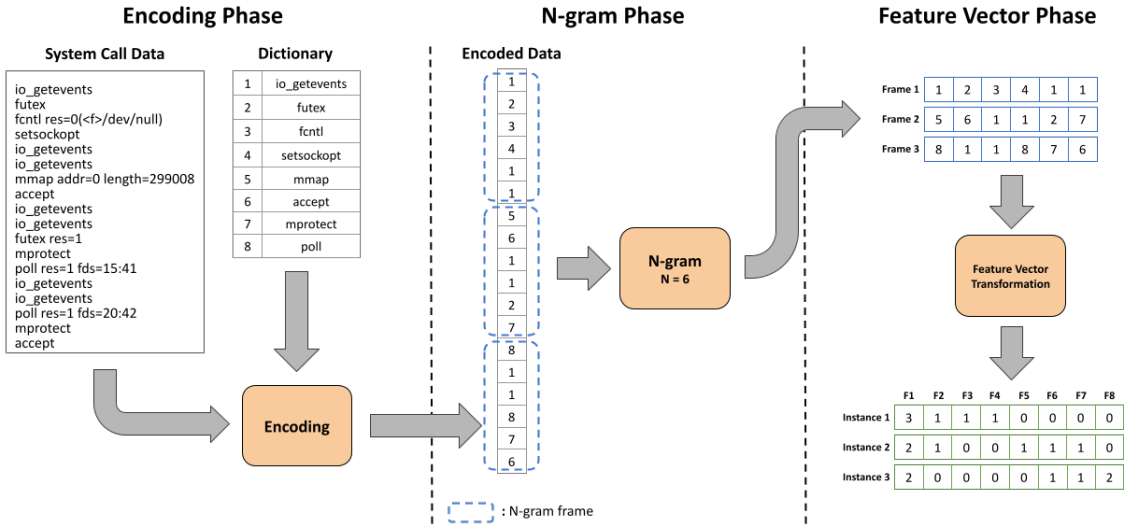

Figure 3: Data pre-processing for the collected system call data.

to collect the next frame and evaluate again if an alert needs to be invoked. This behaviour will continue until there is no frame to complete the size *n* of the window. This sliding window approach intends to increase the number of attack alerts during the timeline attack phase and reduce the number of attack alerts invoked during the pre and post-attack. One challenge of the approach is to determine the ideal window size and the threshold percentage. To determine these variables, we conducted several tests with different results from the test phase of the evaluation procedure.

4.5 Evaluation Procedure

The evaluation procedure described in Section 3 assess ML techniques using different parameters or pre-processing techniques until the results are satisfactory. Although some techniques possess parameters that can apply changes to the trained models, our most significant concern was to test different parameters and pre-processing techniques that could help classify imbalanced data. This concern was born because the training and testing data was imbalanced to the attack class. Table 2 presents the number of instances for each vulnerability dataset and the proportion of attack and normal instances. Based on how imbalanced the data is, we have decided on the test configurations for each technique.

Table 2: Proportion of normal and attack instances in the training and testing data.

Vulnerability	Instances	Normal Instances	Attack Instances
V1	567498	565803 (99.7013%)	1695 (0.2987%)
V2	501380	565803 (99.7013%)	469 (0.0935%)
V3	492055	491276 (99.8417%)	779 (0.1583%)
V4	642561	554446 (86.2869%)	88115 (13.7131%)
V5	565299	562120 (99.4376%)	3179 (0.5624%)

We have decided to use four types of configuration for the DT and RF techniques: *1)* no specific parameters or pre-processing approaches were applied to the ML technique. Thus, only the default specifications were used; *2)* the parameter weight was selected. This parameter will guarantee that the algorithm applies weights to a model's training. The weight is automatically calculated based on the proportion of the classes in the training data. The weight is applied to warn the algorithm that misclassifies an instance from the minority class is worst than one from the majority class. *3)* no specific parameter is used, but an oversampling process is applied to create a new dataset where the number of attack instances is equal to the normal instances. *4)* no specific parameter is used, but an undersampling process is applied to create a new dataset where the number of normal instances is reduced to the same number of attack instances.

We have decided to evaluate the results using different kernel functions for the SVM technique. The four functions available in the scikit-learn and used in the assessment are linear, polynomial, RBF, and sigmoid. Also, the SVM is a technique that can take a long time to train a model if there are many instances. Thus we have decided to apply an undersampling process to reduce the volume of training data for this technique. Table 3 presents the configurations used for the ML Techniques.

Furthermore, some other adjustments needed to be made to the evaluation procedure by the analysis of some preliminary results. The most important one was related to the training data. The testing data was divided into three periods, pre-attack, attack, and post-attack. This division was also made to the training data. However, the training would still use the data from the three periods to train the model without distinction. e noticed that the number of false alarms in the post-attack reached between 30% and 50% of the total data during the initial training and testing. The number of correctly classified attack instances was below 10%. Thus, based on the evidence presented in [9], we decided not to train the models with data from the post-attack. The problem identified is that some vulnerabilities may produce effects that can affect the systems for an extended period that surpass the duration of the attack and go

Table 3: Configurations used for the tests of each supervised technique

Configuration	Decision Tree (DT)	Random Forest (RF)	SVM
Config 1	Default Parameters	Default Parameters	Kernel Linear
Config 2	Weight Parameter	Weight Parameter	Kernel Poly
Config 3	Oversampling	Oversampling	Kernel RBF
Config 4	Undersampling	Undersampling	Kernel Sigmoid

into the post-attack. Thus, training the models with data from the post-attack regarding it as normal data is not a good practice. In this work, we have decided to only use for training the data from the pre-attack and attack, and results have shown an improvement in the classification of attack instances during the attack phase.

Although the earlier adjustment to the training data has improved the results, the high number of false alarms was still a problem for our evaluation. We evaluated that the data collected in $30min$ maybe was not enough to produce a good variety of normal instances. We have decided to include a dataset from the same system in the training data to correct this problem, but it only contains benign data collected throughout $24H$. This addition has dramatically improved the results since it has diminished the number of false alarms pre and post-attack. In addition to earlier measures to improve the quality of the training data, we have also decided to use cross-validation in our evaluation procedure to validate if the model created can generalise the different subsamples of the data. In this preliminary work, we used the cross-validation procedure k-folds to resample our data with the parameter of $k = 10$ for the supervised techniques. The results in the next section already consider the sum of the confusion matrices from the cross-validation process.

5 RESULTS AND DISCUSSION

The evaluation procedure was assessed by two tests, which are described next. *1)* The first test uses the three collections from each vulnerability to train and test the models. *2)* The second test creates a new collection containing information from all the vulnerabilities. This new collection includes two data collections from each vulnerability and the normal data collected over $24H$. The unified collection is used to train the model, and the remaining collection from the vulnerabilities is used to test each vulnerability's model. Following, the results will be analysed by vulnerability, technique, and frame size.

5.1 Vulnerabilities Analysis

The first test verifies the capacity of the ML techniques to create a model that could classify the attack instances correctly from the tested vulnerabilities. Figure 4 presents the results from this first test. These results helped verify how the models detected the attacks exploiting the vulnerabilities. Vulnerability V4 had the best results between the techniques and configurations in this initial test. Its TPR during the attack phase reached a value of 0.98 in most techniques. However, the FPR in pre and post-attack phases were the lowest ranging from 0.05 to 0.014. This result indicates that the models created can detect the attacks for this vulnerability without raising too many false alarms.

On the other hand, for vulnerability V1 in the first test, the TPR for some techniques and configurations are good enough to identify almost 50% of the attack instances. However, the FPR in the pre and post-attack are incredibly high. The vulnerabilities V2 and V3 have similar results to V1 and cannot to detect attacks, at the moment, without creating a large number of false alarms. A fascinating realisation is that not all attack instances need to be detected in the attack phase before the intrusion detection system generates a response. This realisation makes it acceptable to have a low TPR in the attack phase, between 0.2 and 0.3, if the FPR in the pre and post-attack phases are very low, not surpassing 0.10. Another remark from the analysis of the results is that the false alarms will be more frequent during post-attack because the system is still recovering from an intrusion. Thus, for a period after the attack, the expected execution of the system is not correctly active yet.

The second test verifies that adding additional normal instances to the training data will minimise the false alarm ratio in the pre and post-attack. Figure 5 presents the results from this second test. The first noticeable difference between the results of this test and the first one is a reduction in the false alarm ratio of all vulnerabilities. However, this reduction was also observed in the TPR for the vulnerabilities V1, V2, and V3. The evidence provided by the results points towards datasets with few attack instances that cannot train adequate models. Moreover, vulnerabilities V4 and V5 were not negatively affected by this test, keeping similar results to the first test.

5.2 Techniques Analysis

Some remarks are noticeable in the results when considering each technique among the tests. The most noticeable ones are the behaviours of the techniques decision tree and random forest in configuration 4, which performs an undersampling before the training and testing. For this configuration, the FPR results in the first and second tests for the pre-attack and post-attacks are incredibly high compared to the other configurations. For example, in the first test, the decision tree FPR ranged between 0.016 to 0.698, while the other configuration's FPR ranged from 0.10 to 0.330. This event can also be observed in the second test but in a more subtle way. The increase in the FPR is noticeable; however, it does not rise above 0.285 for decision three and 0.236 for the random forest. These results indicate that undersampling for this data is not appropriate because the resulting dataset is too small to contain enough information to classify the instances correctly.

Comparing the decision tree and random forest results, we see a slightly better result for FPR and TPR in the random forest. This result happens to the first test and for the second as well. As the

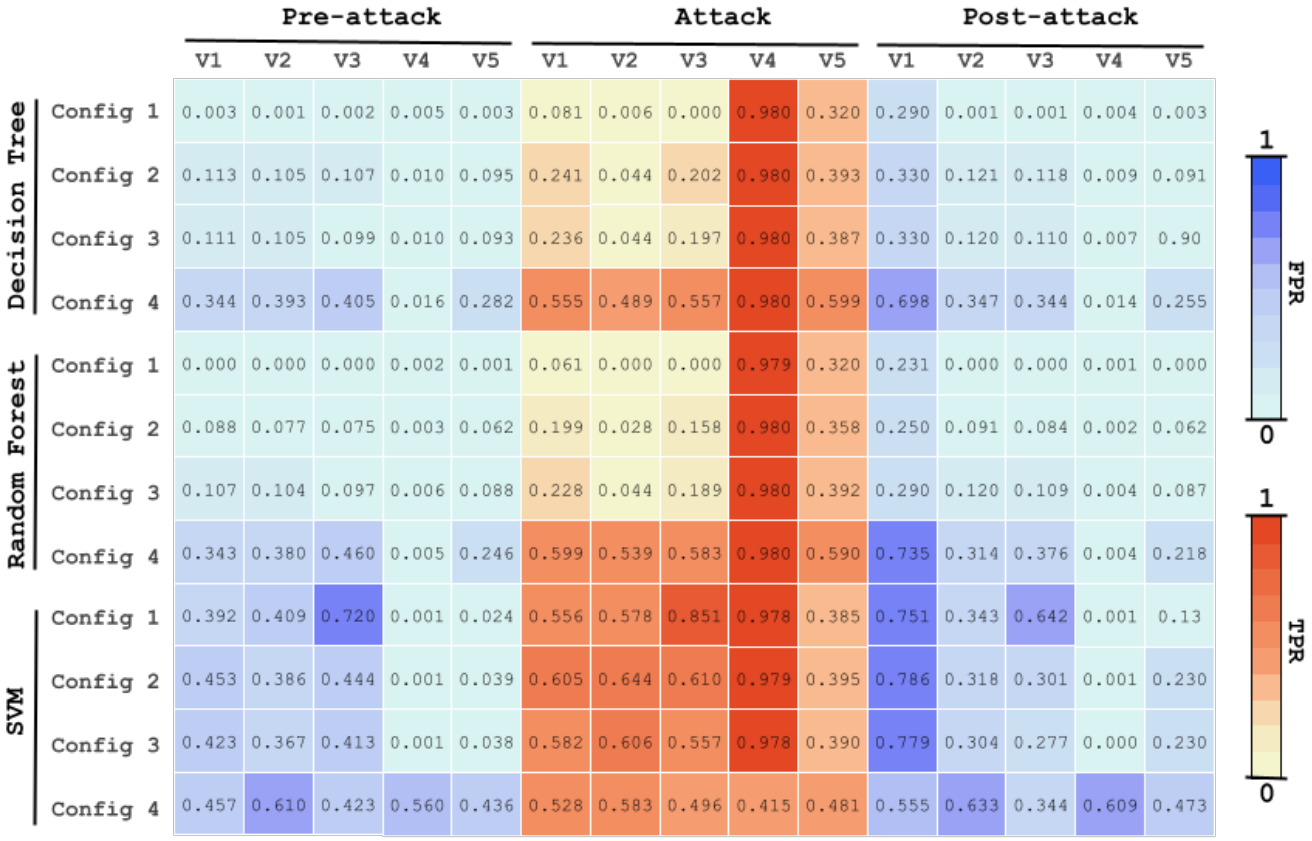

Figure 4: Heatmap of the results from the first test.

random forest uses more than one tree to create the final model, it gains an advantage over a single decision tree. However, it is necessary to investigate if the created model and its creation cost are worthy of the detection gains.

As the configurations used undersampling, the same results as in DT and RF with undersampling were expected. In the first test, the SVM results presented the highest values for FPR from all the techniques. For instance, the post-processing values between SVM configurations 1,2 and 3 were, at their highest, respectively, 0.751, 0.786 and 0.779. However, this tendency is not continued in the second test in which these SVM configurations received results as good as other tested techniques. Exceptionally in the first test, the only divergent result for vulnerabilities V4 and V5 was for the technique SVM with kernel Sigmoid, where its FPR ranged from 0.560 to 0.609. Again in the second test, the technique SVM with Config 4 results were poor. The results show that the kernel Sigmoid cannot handle the scenario since its results are creating a high false alarm rate and minimising the TPR. Furthermore, in Figure 5, the technique and configuration were not affected as strongly as the other kernels by using a large volume of normal data.

5.3 Frames Size Analysis

Our decision to use $n = 50$ as our default frame size was based on the analysis of the expanded results of the first test presented in Table 4 for the decision tree, Table 5 for the random forest and Table 6 for the SVM results. These tables contain the values for TPR and FPR during the execution phases to detect attacks through vulnerability V1. In Table 4, we can see the FPR values for pre-attack are getting lower as we increase the size of the frames for Configurations 1,2, and 3. However, for the post-processing FPR, the values commonly increase until frame size $n = 400$ when the value decreases. This behaviour is also typical of the TPR results in the attack phase. This information helped us understand the limitation on the size of the frame. More oversized frames have less information available to the model since it evaluates a feature vector, and their internal sequence of system calls is not analysed. This situation can increase the number of identical frames in different phases, making it difficult to establish a rule to classify them.

Thus, it is possible to understand how the frame size $n = 50$ was chosen by analysing the other results. In table 5, the same behaviour is identified. Frame size $n = 50$ is the only one where the TPR is higher than other sizes, and the FPR during pre and post-attacks is the lowest possible. Thus as we intend to reward techniques and configurations with high attack phase TPR and low pre and post-attack FPR, $n = 50$ is the best choice.

In addition, some remarks can be retrieved from configuration 4 of Tables 4 and 5. It is possible to see that independent of the frame size, the values for all phases in configuration 4 are higher

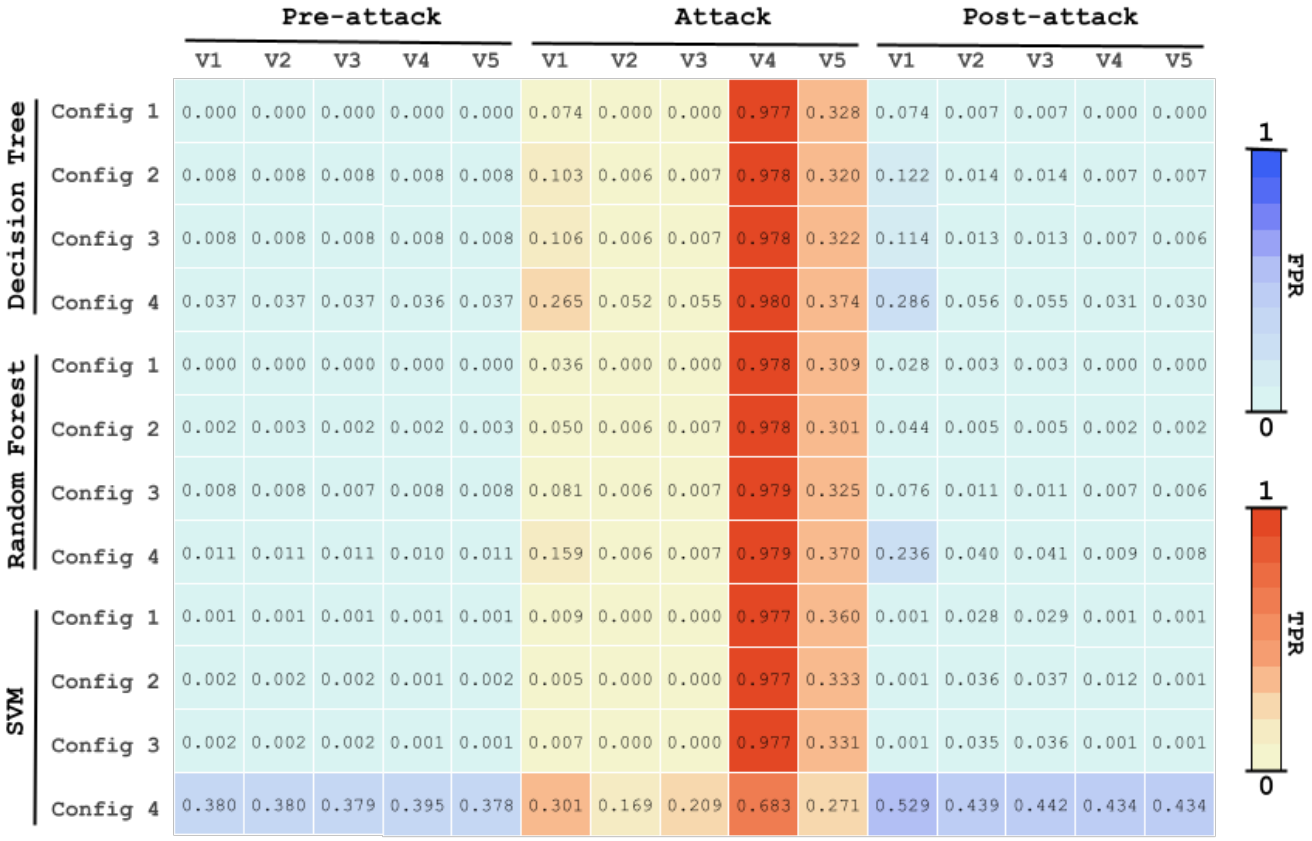

Figure 5: Heatmap of the results from the second test.

than any other configuration. Also, the results for this configuration have low variance between them, indicating that even when increasing the frame size, the dataset formed by the undersampling does not provide enough information for the correct classification of instances. Lastly, the results from Table 6 are not suitable for a frame size evaluation since the FPR is very high. These results are an effect of the undersampling that they are submitted. However, as the second test results suggest, undersampling can be used for this technique as long we provide a sizable dataset with enough normal data and enough representation of the attack instances.

5.4 Post-processing Results

Based on the results of the second test in Figure 5, we evaluated how a sliding window post-processing approach could help improve the test phase results. Our idea was to minimise the number of false alarms during the pre and post-attack and increase the number of attack alerts during the attack period. For this analysis, we used the results from the Decision Tree with the undersampling configuration in test two as the post-processing object. In our tests, this pair produces the worst classification results because the subsample created is too small to represent all system call data, creating many false alarms during the pre and post-attack. In addition, the tests were used to gather information about the ideal values for the window size and the alert threshold because they are important to obtain relevant results from the post-processing approach. The window sizes tested were $n = \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50\}$ and the threshold values were $p = \{5\%, 10\%, 15\%, 20\%\}$. Our tests were executed on all five vulnerabilities. However, we selected V5, V4, and V2 results to discuss since they encapsulate all results' characteristics.

Starting with vulnerability V5, the results from earlier sections show that supervised techniques could correctly classify its attack frames in a fair proportion without throwing false alarms. Post-processing results verified that by increasing the threshold value, the number of attack alerts in pre and post-attack was minimised from 30% to below 1%. Also, as the window size increases, the proportion of attack alerts is better during the attack phase, with a TPR variation average of 0.2. The vulnerability V4 results show us a scenario in which attack alerts in the attack phase gain little from the post-processing approach. V4 is a vulnerability in which our tests could secure a high TPR and an exceptionally low number of false alarms. The only scenario where the post-processing approach helps is when undersampling is used. In the results, we verified that to a higher threshold. The attack alerts are diminished during pre and post-processing attacks. However, the attack alerts during the attack phase do not show any differences since the classification of frames could secure the best scenario in the attack phase.

Table 4: Results from different frames sizes for the Decision Tree in Vulnerability V1

	Pre-attack (FPR)				Attack (TPR)				Post-attack (FPR)			
Frame Size	Config1	Config2	Config3	Config4	Config1	Config2	Config3	Config4	Config1	Config2	Config3	Config4
35	0.002	0.185	0.191	0.361	0.062	0.245	0.243	0.507	0.170	0.407	0.404	0.630
50	0.003	0.113	0.111	0.344	0.081	0.241	0.236	0.555	0.290	0.330	0.330	0.698
100	0.004	0.053	0.052	0.350	0.117	0.169	0.178	0.574	0.456	0.432	0.432	0.746
200	0.006	0.015	0.019	0.386	0.127	0.128	0.133	0.583	0.508	0.518	0.528	0.791
400	0.006	0.006	0.006	0.346	0.103	0.123	0.123	0.529	0.484	0.403	0.409	0.796

Table 5: Results from different frames sizes for the Random Forest in Vulnerability V1

	Pre-attack (FPR)				Attack (TPR)				Post-attack (FPR)			
Frame Size	Config1	Config2	Config3	Config4	Config1	Config2	Config3	Config4	Config1	Config2	Config3	Config4
35	0.000	0.156	0.189	0.381	0.049	0.199	0.237	0.573	0.193	0.352	0.395	0.694
50	0.000	0.088	0.107	0.343	0.061	0.199	0.228	0.599	0.231	0.250	0.290	0.735
100	0.000	0.030	0.048	0.341	0.101	0.130	0.171	0.599	0.418	0.309	0.368	0.786
200	0.000	0.008	0.0013	0.326	0.104	0.030	0.056	0.563	0.457	0.190	0.273	0.814
400	0.116	0.001	0.001	0.314	0.089	0.023	0.060	0.624	0.485	0.157	0.197	0.746

Table 6: Results from different frames sizes for the Support Vector Machine (SVM) in Vulnerability V1

	Pre-attack (FPR)				Attack (TPR)				Post-attack (FPR)			
Frame Size	Config1	Config2	Config3	Config4	Config1	Config2	Config3	Config4	Config1	Config2	Config3	Config4
35	0.416	0.422	0.404	0.445	0.571	0.586	0.574	0.546	0.712	0.702	0.699	0.543
50	0.392	0.453	0.423	0.457	0.556	0.605	0.585	0.528	0.751	0.786	0.779	0.555
100	0.188	0.303	0.256	0.395	0.462	0.531	0.502	0.533	0.774	0.803	0.795	0.623
200	0.202	0.281	0.234	0.415	0.480	0.502	0.478	0.576	0.805	0.814	0.750	0.801
400	0.189	0.551	0.464	0.856	0.515	0.686	0.639	0.990	0.841	0.891	0.855	0.854

Lastly, in vulnerability V2, no ML technique created a model that could classify the test frames correctly. Thus, the three phases are affected equally by the process when we use the post-processing approach. The number of attack alerts is minimised, including the attack phase in which we want to increase the attack alerts instead. In this case, it is possible to evaluate that if the ML techniques cannot create an adequate model to classify the fessentialrectly, the post-processing approach cannot help. The results for vulnerabilities V1 and V3 are highly identical to V2 since the models could not classify the attacks. Using the results from V5, where the post-processing approach was best executed, we evaluated that the ideal threshold would be 15% with a window size between 30 to 45.

5.5 Threats to Validity

Although our various experiments to identify the efficacy of intrusion detection based on machine learning techniques in a microservice scenario resulted in respectable results, some aspects of this work may be challenging an overall recognition of the possibilities for ML-based techniques.

First, ML possesses many different techniques that can be used in classification. However, our study has focused on supervised techniques, which are just a group inside the pool of possible techniques. Also, this study was restricted to three techniques, and even with the results being promising overall, more techniques need to be evaluated to strengthen the results. Additionally, many parameters are available to each technique and can significantly change the results. In this study, the parameters were used only to deal with imbalanced data problems, which were necessary at the time. Still, expanding the use of these parameters is ideal for understanding how each technique deals with attacks through different system vulnerabilities.

The training and testing data also provide some limitations to the study. Since vulnerabilities V1, V2, and V3 had so few attack frames, do these vulnerabilities possess some characteristics that difficult the detection of attacks performed through them, or the data available was not enough to train a suitable model. Also, using feature vectors as our data representation may have hindered the gathering of more information among the system calls. This choice was necessary to allow us to test different sizes of frames sizes using the setup and resources available but reduced the information at hand for the classification.

6 CONCLUSION AND FUTURE WORK

This work evaluated machine learning techniques for intrusion detection in a microservice scenario. Our experiment was able to verify if machine learning techniques would be capable of detecting intrusions. Although some intrusions were not correctly detected, two had good detection results and low false alarms. This result indicates that the evaluation procedure can assess machine learning techniques for our microservice scenario. However, it also provides

insights into the quality and quantity of data, reinforcing the need to expand the testing to other techniques with different classification approaches, such as distance/based techniques.

Although few parameters from the techniques were tested, they were enough to show the changes between different configurations and provide information for future technique profiling. Also, the data pre-processing approaches used to refine the training and testing were essential in obtaining favourable results. They will be further explored to verify how it impacts the execution of techniques. In addition, the post-processing approach developed to analyse classified data from models has provided us with details about the limits to improve results. Thus, providing us with more information to adjust our evaluation, which may need more metrics than the TPR and FPR.

Our next steps will be to expand the ML techniques, attacks, and parameters tested in the evaluation, for instance, distance-based techniques. However, we must extend the research into data representation for system calls to use these techniques. First, changing from feature vectors to another format that can maintain the sequence in the frames, and later proposing embedding approaches specifically for system calls to maintain their information and connections. Also, based on the studies with intrusion detection, tolerance for microservice scenarios could be explored using integrated approaches with intrusion detection based on ML techniques.

ACKNOWLEDGMENTS

This work was partially funded by FCT grant 2022.11551.BD. This work has been partially supported by the project AIDA - Adaptive, Intelligent and Distributed Assurance Platform (reference POCI-01-0247-FEDER-045907) leading to this work is co-financed by the ERDF and COMPETE 2020 and by the FCT under CMU Portugal. It is also partially supported bt the FCT – Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit – UIDB/00326/2020 or project code UIDP/00326/2020.

REFERENCES

[1] Rebecca Gurley Bace. 2000. *Intrusion detection.* Sams Publishing.
[2] Anna L Buczak and Erhan Guven. 2015. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials* 18, 2 (2015), 1153–1176.
[3] Marcos Cavalcanti, Pedro Inacio, and Mario 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*. 1–9.
[4] Nadia Chaabouni, Mohamed Mosbah, Akka Zemmari, and Cyrille Sauvignac. 2020. A OneM2M Intrusion Detection and Prevention System based on Edge Machine Learning. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 1–7.
[5] Clarence Chio and David Freeman. 2018. *Machine learning and security: Protecting systems with data and algorithms.* " O'Reilly Media, Inc.".
[6] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering* (2017), 195–216.
[7] Angelo Feraudo, Poonam Yadav, Vadim Safronov, Diana Andreea Popescu, Richard Mortier, Shiqiang Wang, Paolo Bellavista, and Jon Crowcroft. 2020. CoLearn: Enabling federated learning in MUD-compliant IoT edge networks. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*. 25–30.
[8] José Flora. 2020. Improving the Security of Microservice Systems by Detecting and Tolerating Intrusions. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. IEEE, 131–134.
[9] José Flora, Paulo Gonçalves, and Nuno Antunes. 2020. Using attack injection to evaluate intrusion detection effectiveness in container-based systems. In *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 60–69.
[10] Yasir Hamid, M Sugumaran, and Ludovic Journaux. 2016. Machine learning techniques for intrusion detection: a comparative analysis. In *Proceedings of the International Conference on Informatics and Analytics*. 1–6.
[11] Abdelhakim Hannousse and Salima Yahiouche. 2021. Securing microservices and microservice architectures: A systematic mapping study. *Computer Science Review* 41 (2021), 100415.
[12] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35.
[13] Rupesh Raj Karn, Prabhakar Kudva, Hai Huang, Sahil Suneja, and Ibrahim M Elfadel. 2020. Cryptomining Detection in Container Clouds Using System Calls and Explainable Machine Learning. *IEEE Transactions on Parallel and Distributed Systems* 32, 3 (2020), 674–691.
[14] Ansam Khraisat, Iqbal Gondal, Peter Vamplew, and Joarder Kamruzzaman. 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity* 2, 1 (2019), 1–22.
[15] Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and Christof Ebert. 2018. Microservices. *IEEE Software* 35, 3 (2018), 96–100.
[16] Ming Liu, Zhi Xue, Xianghua Xu, Changmin Zhong, and Jinjun Chen. 2018. Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–36.
[17] Nuno Mateus-Coelho, Manuela Cruz-Cunha, and Luis Gonzaga Ferreira. 2021. Security in Microservices Architectures. *Procedia Computer Science* 181 (2021), 1225–1236.
[18] Marc-Oliver Pahl and François-Xavier Aubet. 2018. All eyes on you: Distributed Multi-Dimensional IoT microservice anomaly detection. In *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE, 72–80.
[19] Anelis Pereira-Vale, Gastón Márquez, Hernán Astudillo, and Eduardo B Fernandez. 2019. Security mechanisms used in microservices-based systems: A systematic mapping. In *2019 XLV Latin American Computing Conference (CLEI)*. IEEE, 01–10.
[20] Bhawana Sharma, Lokesh Sharma, and Chhagan Lal. 2019. Anomaly detection techniques using deep learning in iot: A survey. In *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*. IEEE, 146–149.
[21] Peiying Tao, Zhe Sun, and Zhixin Sun. 2018. An improved intrusion detection algorithm based on GA and SVM. *Ieee Access* 6 (2018), 13624–13631.
[22] Han Wang, Luis Barriga, Arash Vahidi, and Shahid Raza. 2019. Machine Learning for Security at the IoT Edge-A Feasibility Study. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW)*. IEEE, 7–12.
[23] Huiwen Wang, Jie Gu, and Shanshan Wang. 2017. An effective intrusion detection framework based on SVM with feature augmentation. *Knowledge-Based Systems* 136 (2017), 130–139.
[24] Yu Wang, Weizhi Meng, Wenjuan Li, Zhe Liu, Yang Liu, and Hanxiao Xue. 2019. Adaptive machine learning-based alarm reduction via edge computing for distributed intrusion detection systems. *Concurrency and Computation: Practice and Experience* 31, 19 (2019), e5101.
[25] Dongjin Yu, Yike Jin, Yuqun Zhang, and Xi Zheng. 2019. A survey on security issues in services communication of Microservices-enabled fog applications. *Concurrency and Computation: Practice and Experience* 31, 22 (2019), e4436.
[26] Olaf Zimmermann. 2017. Microservices tenets. *Computer Science-Research and Development* 32, 3 (2017), 301–310.