



## UvA-DARE (Digital Academic Repository)

### A real-time intrusion detection system based on OC-SVM for containerized applications

Zhang, L.; Cushing, R.; de Laat, C.; Grosso, P.

**DOI**

[10.1109/CSE53436.2021.00029](https://doi.org/10.1109/CSE53436.2021.00029)

**Publication date**

2021

**Document Version**

Final published version

**Published in**

Proceedings, 2021 IEEE 24th International Conference on Computational Science and Engineering

**License**

Article 25fa Dutch Copyright Act (<https://www.openaccess.nl/en/policies/open-access-in-dutch-copyright-law-taverne-amendment>)

[Link to publication](#)

**Citation for published version (APA):**

Zhang, L., Cushing, R., de Laat, C., & Grosso, P. (2021). A real-time intrusion detection system based on OC-SVM for containerized applications. In A. Hawbani, Z. Li, & A. Muthanna (Eds.), *Proceedings, 2021 IEEE 24th International Conference on Computational Science and Engineering: CSE 2021 : Shenyang, China, 20-22 October 2021* (pp. 138-145). IEEE Computer Society. <https://doi.org/10.1109/CSE53436.2021.00029>

**General rights**

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

**Disclaimer/Complaints regulations**

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible. University of Amsterdam (<https://dare.uva.nl>)

# A real-time intrusion detection system based on OC-SVM for containerized applications

1<sup>st</sup> Lu Zhang

*MultiScale Networked Systems (MNS)*  
University of Amsterdam  
Amsterdam, The Netherlands  
l.zhang2@uva.nl

2<sup>nd</sup> Reginald Cushing

*MultiScale Networked Systems (MNS)*  
University of Amsterdam  
Amsterdam, The Netherlands  
r.s.cushing@uva.nl

3<sup>rd</sup> Cees de Laat

*Complex Cyber Infrastructure (CCI)*  
University of Amsterdam  
Amsterdam, The Netherlands  
C.T.A.M.deLaat@uva.nl

4<sup>th</sup> Paola Grosso

*MultiScale Networked Systems (MNS)*  
University of Amsterdam  
Amsterdam, The Netherlands  
p.grosso@uva.nl

**Abstract**—A Digital Data Marketplace (DDM) is a digital infrastructure to facilitate policy-governed data sharing in a secure and trustworthy manner with container-based virtualization technologies. An intrusion detection systems (IDS) is essential to enforce the policies. We propose a real-time intrusion detection system that monitors and analyzes the Linux-kernel system calls of a running container. We adopt the One-Class Support Vector Machine (OC-SVM) to detect anomalies. The training data of the OC-SVM algorithm is collected and sanitized in a secure environment. We evaluate the detection capability of our proposed system against modern attacks, e.g. Machine Learning (ML) adversarial attacks, with a customized attack dataset. In addition, we investigate the influence of various feature extraction methods, kernel functions and segmentation length with four metrics. Our experimental results show that we can achieve a low FPR, with a worst case of 0.12, and a TPR of 1 for most attacks, when we adopt the *term-frequency* feature extraction method and we choose segmentation length of 30000. Furthermore, the optimal kernel functions depend on the concrete application being examined.

**Index Terms**—Data sharing, Intrusion Detection, Anomaly Detection, OC-SVM

## I. INTRODUCTION

A Digital Data Marketplace (DDM) is a digital infrastructure that facilitates secure data exchange and federation. For instance, different DDM parties may want to gather their local data together and run a machine learning (ML) algorithm on their joint data, so that they can gain benefits from a more accurate prediction model. Those parties could be competing, so they concern about the confidentiality of their data and whether the computing result is trustworthy [1], [2]. In a DDM, there is a unique identifier for each data and compute object. The parties agree on permissible actions on specific data and compute objects and express them into a policy [3]. The compute objects are containerized for better portability. Containers are operating system level virtualization abstractions. A container image is a lightweight, executable package including source code, program runtime and libraries [4].

The DDM infrastructure implements policy enforcement components to mitigate possible vulnerabilities faced by such data exchange applications. For example, most ML models are vulnerable to adversarial attacks, which degrades the model performance by modifying training samples in the runtime [5]. Container technology also creates new vulnerabilities, such as container escalation attack [6]. Preventive countermeasures must be implemented as the first line of defense. However, attackers will keep developing novel techniques to bypass the existing security mechanisms. Hence, a real-time intrusion detection system (IDS) is essential for such digital infrastructures.

The system call is an interface between an application and Linux kernel and is a widely used monitoring metric for intrusion detection. System calls are used by user-level applications for various reasons including file management, process control, device management and communication. These system calls give us a way to monitor what user-level applications are doing. An IDS can be mainly classified into two categories, namely, signature-based and anomaly-based. The signature-based IDS detects attacks by matching the monitoring metrics with existing patterns. Consequently, it can not identify zero-day attacks. An anomaly-based IDS learns a profile describing normal behaviors and flag a potential attack if a sufficient deviation from the normal profile occurs [7].

We propose a hybrid real-time intrusion detection system with system calls generated by a running container. We adopt One Class Support Vector Machine (OC-SVM) as the anomaly detection algorithm due to its capability of dealing with complex non-linear problems. To detect malicious behaviors in a real-time manner, the streaming system calls are separated into segments before being mapped into feature vectors. Then we apply the signature-based methodology to reduce false alarms. For each compute object, an anomaly detection algorithm is trained if it was used for the first time. Adapting to the dynamic characteristics of the application behavior, the anomaly detection algorithm is retrained whenever new data

is available. It is vital to ensure that both training and retraining data are attack-free. In our proposed system, the training data is collected in a secured environment and retraining data is analyzed and sanitized.

We also evaluate how the OC-SVM algorithm works for detecting anomalies with system call traces. The performance highly depends on the statistical distribution of the attack traces and modern attacks are more difficult to distinguish [8]. We construct a new dataset including system call traces of modern container-specific attacks and adversarial ML attacks. Three numeric metrics are measured to evaluate the performance, namely, True Positive Rate (TPR), False Positive Rate (FPR) and Area under the ROC curve (AUC). We also investigate how the different feature extraction methods, kernel functions, segmentation lengths, influence the IDS performance.

In Section II we compare our system and methodology with existing ones. We present the full architecture of our system in Section III. The details of the *Detection Engine (DE)* are explained in Section IV. Section V and Section VI present the constructed dataset and the experimental setups. Our results, analysis and conclusions are discussed in Section VII, Section VIII and Section IX.

## II. RELATED WORK

There is a large amount of literatures using system calls to detect potential malicious behaviors.

The work in [9] was one of the first to use system call traces to characterize the behaviors of a running program. It builds a dataset of normal behaviors with a fixed length system call subsequences. After this, a test trace is identified as anomalous if the number of mismatch subsequences exceeds a user-defined threshold. The problem with this approach is that it lacks generalization and consumes huge storage capacity. The work in [10] uses system call to model the normal profiles as a Hidden Markov Chain (HMM), but tuning parameters of an HMM is extremely time consuming.

Recently, machine learning techniques have started to also be widely used for building anomaly-based IDS. The work in [11] proposes an IDS with K-nearest-neighbor (KNN) algorithm and evaluates the performance with the DARPA dataset. This approach does not require a separate profile for each program but the detection accuracy for novel attacks is only 75%. The work in [12] uses ngrams as feature vectors and compares multiple learning algorithms, namely, Support Vector Machine (SVM), Multilayer Perceptron (MLP) and Naive Bayes. They evaluate the performance with the ADFA-LD public dataset and observed that SVM outperforms the other two. There are also work that similarly to ours use OC-SVM as underlying technique. [13] evaluates the performance of the OC-SVM algorithm with different kernel functions also with the ADFA-LD public dataset. The work shows that OC-SVM can gain satisfactory performance with low computational cost. [14] proposes an OC-SVM based anomaly detection system using frequency distribution of various length

n-grams as the feature vector. They also conclude that OC-SVM outperforms sequential anomaly detection models with the ADFA-LD dataset. However, both works fail to address the real time requirement of modern IDS systems, which we do cover in our work.

## III. SYSTEM DESCRIPTION

Figure 1 describes the architecture of our proposed real-time intrusion detection system based on OC-SVM. In general, the training stage is conducted offline in a secured environment, a *authorized party*, in a centralized manner and then distributed to the *endpoint execution platform*, for real time anomaly detection. The *centralized authorized party* consists of an *Initial Training* module, an *Integrity Verification and Retraining* module and a *Model Database*. The *Endpoint Execution Platform* contains a *System Call Monitoring* module and a *Decision Engine*.

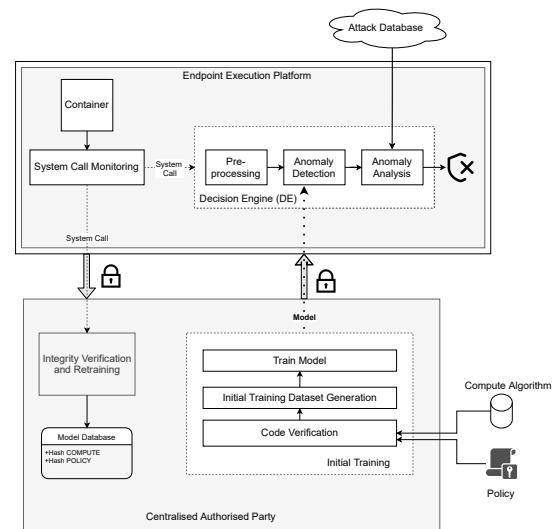


Fig. 1: The architecture of the intrusion detection system.

Before delegating a data exchange application to a DDM, the collaborating parties first agree on a policy describing permissible actions on the data and compute objects. The policy and compute objects are sent to a trustworthy *authorized party*. The security experts check the source code manually and verify whether it complies with the policy. Then the *authorized party* generates the initial training data and trains the anomaly detection algorithm. This ensures that the training data is clean and not contaminated by adversaries. Both the pre-trained detection model and verified compute objects are sent to the *endpoint execution platform* via a secured communication channel. The *authorized party* signs and encrypts the pre-trained detection model and the compute object, normally a container image, with the public key of the *endpoint execution platform* to ensure integrity.

On the *endpoint execution platform*, the containerized compute object may perform operations on the data objects agreed in the policy. The *system call monitoring* module gathers the

system call traces with Sysdig. It sends the streaming system calls to the *Detection Engine* module. This module detects anomalies with the pre-trained OC-SVM model and decides whether it is necessary to apply countermeasures. The details of this module will be discussed in Section IV.

In the meantime, the *System Call Monitoring* module also sends the system calls back to the *authorized party* for model retraining if needed. The behaviors of some applications are dynamic in nature, so it is necessary to retrain the model periodically. The *Integrity Verification and Retraining* module checks the integrity of the received system call traces and verifies whether they are attack free. The same anomaly detection model, with exactly the same parameters, is run in parallel with the received streaming system call traces in the *authorized party*. The receiving system calls are recognized as attack-free if the alarm rate is close to the recorded FPR value of this model in the initial training. Then these system call traces are allowed to retrain the model. We adopt this mechanism to reduce the likelihood that the model is retrained with contaminated samples.

#### IV. DETECTION ENGINE

The Detection Engine (DE) is comprised of three components: a Pre-Processing Module, an Anomaly Detection module and an Anomaly Analysis model.

##### A. Pre-processing Module

When the container runs, the *System Call Monitoring* module captures the system calls and passes them to the *Pre-processing* module. This divides the streaming system calls into segments with a fixed window size: this is needed because the inputs of the classic ML algorithms are fixed length vectors. Furthermore this segmentation allows us to perform our detection while the systems calls are coming in real time.

The Pre-Processing module also maps these segments system call traces into vectors in the feature space. In our work we considered three feature extraction methods, namely *tf*, *tf-idf* and *ngram*. We will describe them in more detail in Section VI-B

##### B. Anomaly Detection Module

A pre-trained IDS learning algorithm is running in the *Anomaly Detection* module and determines whether an input feature vector is anomalous or not. Here we adopted One-class Support Vector Machine (OC-SVM) as the IDS learning algorithm. Our choice stems from the fact that OC-SVM is good at dealing with complex non-linear problems. This results in it being widely used for intrusion detection systems [13].

The general idea of an SVM algorithm is to find a hyperplane that separates the normal and abnormal data points with a maximized geometry margin by solving an optimization problem. It maps the input data points into a new feature space of higher or even infinite dimensions with kernel functions. Therefore, the original linearly non-separable data patterns may be converted into, with high likelihood, linearly separable patterns in the high dimensional space [15].

Similar to standard SVM, the OC-SVM also aims to find a decision boundary with a maximum geometry margin [16]. However, it is an unsupervised learning algorithm and does not require any labeled training data. This is suitable for usage in anomaly detection, where training datasets are normally unbalanced. The OC-SVM algorithm considers the anomalous data points to be close to the origin, while the normal data points are far from the origin. It uses a spherical decision boundary instead of a plane in the higher-dimension feature space [17]. The algorithm aims to find a sphere with minimal volume that contains the most normal data points. The objective function is:

$$\begin{aligned} \min_{R, \bar{a}} R^2 + C \sum_{i=1}^N \xi_i \\ \text{subject to :} \\ \|\Phi(X_i) - \bar{a}\| \leq R^2 + \xi_i, \quad \text{for } i = 1, 2, \dots, N \\ \xi_i \geq 0, \quad \text{for } i = 1, 2, \dots, N \end{aligned} \quad (1)$$

The spherical decision boundary is characterized with its center  $\bar{a}$  and the radius  $R$ .  $X_i$  is the  $i$ th training data in input space  $I$  and  $N$  is the total number of training samples.  $\Phi()$  denotes a feature map from the input space  $I$  to a high-dimensional feature space  $F$ .  $\xi_i$  is a slack variable to prevent over-fitting from some noisy data points by creating a soft margin. It allows some data points to lie within the margin.  $C$  is a constant to determine the trade-off between the sphere volume and the number of data points it can hold.

A kernel function is defined as:

$$K(X_i, X_j) = \Phi(X_i) \cdot \Phi(X_j)^T \quad (2)$$

Given two data points  $X_i, X_j$ , the output of the kernel function is the dot product of their mapping in new space  $F$ . As the decision boundary of an SVM algorithm only relies on the dot product in feature space  $F$ , the explicit projection is not necessary. In our work, we chose two popular kernels, linear and Gaussian, to evaluate the performance of our system. The details will be discussed in section VI-C.

##### C. Anomaly Analysis Module

The *Anomaly Analysis* module analyses the anomalies identified by the *Anomaly Detection* module and provides information to the security experts determining whether to take countermeasures or not. For an input segment tracefile, which is flagged as an anomaly, this module conducts the following operations:

- Match the system call sequences with existing attack database;
- Check the neighboring segments of system call traces and determine whether this is a standalone anomaly or not;

An identified anomaly is recognized as standalone if all the data points in the neighboring area ( $N_s$  points before and  $N_s$  points after) are identified as normal.  $N_s$  is an adjustable parameter to define the range of the neighboring area of a data point. The *Anomaly Analysis* module assigns each anomaly

a level based on the outcomes of the above operations. An anomaly is marked as 'High' if the traces match any existing attack in the database. An anomaly is marked as "Low" if it is a standalone point. The remains are marked as "Medium". The detailed matching approach and its performance has been explained in our previous paper [18]. The effectiveness of recognizing standalone points will be discussed in Section VIII.

## V. EXPERIMENTAL DATASET CONSTRUCTION

The emerging microservices pose many challenges for real-time intrusion detection systems, due to their highly dynamic natures. Also, the performance of IDS learning algorithms highly depends on the statistical properties of the training dataset. When starting our evaluation we found out that there are currently no public databases containing system call traces of container-specific applications or attacks. We therefore set out to construct our own training dataset, in order to validate how *oc-svm* works for detecting modern attacks for containerized applications.

To tailor our work to the DDMs we first identified the most typical applications expected to run in such environments: databases services and training machine learning models are the most likely type of use cases. The former are examples of dynamic applications with many users interacting with the system; the latter are more static applications that do not have many users involved and have a more constant execution path every time. For each one of them we implement an attack with penetration tools which provides us with a suitable dataset.

### A. Dynamic Applications: CouchDB and MongoDB

As example of dynamic applications we select two NoSQLMap databases, CouchDB and MongoDB, to investigate the performance of the proposed intrusion detection system.

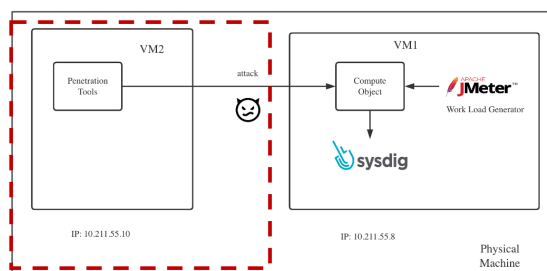


Fig. 2: Platform for generation of normal and abnormal traces for dynamic applications (CouchDB and MongoDB).

Figure 2 shows the experimental platform we built for generating normal and abnormal system call traces. To avoid the inferences of other user activities, we set up a Docker container running CouchDB or MongoDB server in a virtual machine (VM1). Sysdig is implemented in the kernel space to collect system call traces generated by the running container.

To simulate the dynamic behaviors of real-world database users, we send requests to the container from the same virtual machine (VM1). This is conducted with Apache JMeter, which is an open-source workload generation tool. For abnormal traces, we conduct attacks with exploitation tools, e.g Nmap and Metasploit, in another virtual machine (VM2).

For CouchDB, we use the HTTP sampler of JMeter. This sampler enables choosing the proper HTTP traffic types, e.g GET or POST. For MongoDB we use JSR 223 Sampler of JMeter to generate the traffic. Web requests are sent to the server to induce database operations. For each application, there are two threads in JMeter, thread 1 and thread 2, which send requests to the containerized server simultaneously. Thread 1 includes 100 users, who perform operations of inquiring documents and basic information in different databases. Thread 2 includes 3 users, who perform operations including updating, creating and deleting databases. While JMeter is generating dynamic traffic, Sysdig monitors the system calls of the server container, which is used as the normal traces of the application. Table I shows the information of the performed attacks and tracefile sizes for the two dynamic applications.

### B. Static Application: Machine Learning (ML) Applications

Another common application for DDM is to train a ML learning model with data from multiple parties. We chose image classification with Concurrent Neural Networks (CNN) with MINST dataset to validate the performance of our anomaly detection system.

Recent research shows that deep learning algorithms are vulnerable to adversarial attacks. [19]. This attack generates adversarial training samples in the runtime. The adversarial samples are nearly unnoticeable for humans but can fool the model with high confidence. There are a number of adversarial sample generation approaches in the literature [20]. Concretely, we adopted the Projected Gradient Descent (PGD), Basic Iterative Method (BIM), Carlini and Wagner (CW), Fast Adaptive Boundary (FAB), multiple steps fast gradient symbol method (MIFGSM), PGDDL, Square and TPGD method to generate adversarial samples [21].

The ML algorithms is encapsulated with a Docker container. For normal traces of the application, Sysdig collects system calls for the entire training stage. For abnormal traces, Sysdig gathers system calls when the block of code, which generates the adversarial samples, is executed. Table II shows the information of the performed attacks and tracefile size for the image classification application.

## VI. EXPERIMENTAL DESIGN

In this section, we implement the proposed architecture and evaluate its performance. The key question we want to answer is: *How does the OC-SVM based DE perform for detecting modern attacks?*

More precisely, we want to investigate the influence of different feature extraction methods, different kernel functions and different segmentation lengths.

TABLE I: Applications and attacks of the constructed dataset for the two dynamic applications

Application	Attack	Exploitation Tool	Jmeter Sampler	Number of System Call Symbols	
				Normal	Abnormal
CouchDB	Container Escalation + Execute Arbitrary Code	Metasploit	HTTP Sampler	7458046	5199817
MongoDB	Brute Force	Nmap	JSR 223 Sampler	40463150	4449052

TABLE II: Applications and attacks of the constructed dataset for the ML static application

Application	Attacks	Exploitation Tool	Number of System Call Symbols	
			Normal	Abnormal
Image Classification	Adversarial ML: PGD	Proof of Concept	2798258	4360000
	Adversarial ML: BIM			960967
	Adversarial ML: CW			4728113
	Adversarial ML: FAB			1394676
	Adversarial ML: MIFGSM			118007
	Adversarial ML: PGDDLRL			466024
	Adversarial ML: Square			182813
	Adversarial ML: TPGD			87884

The experiments are conducted with the dataset discussed in Section V. OC-SVM is an unsupervised learning algorithm; the model is trained with only normal data and tested with both normal and abnormal data. To avoid over-fitting, in our experiments we use K-fold ( $K = 10$ ) cross validation. We first shuffle the normal data points randomly and split them into  $K$  folds. For each interaction  $k \in \llbracket 1, K \rrbracket$ , one fold of the normal data points and all the abnormal data points are used for testing. The remaining 9 folds are used for training the model. After  $K$  interactions, every fold has been used once for testing. The final value of the evaluation metric is the average of  $K$  values [22].

We deployed our experiments in a VM equipped with 4 CPU cores at 2.9 GHz and 16 GB memory. The OS is Ubuntu 18.04 LTS, kernel 4.15.0.

#### A. Segmentation Length

As already discussed in Section III, the streaming system calls are divided into segments to achieve detection results in the real time. The window size that segments the trace is called *segmentation length* and denoted as  $L_s$ . Hence a trace of  $L$  system call symbols can be split into  $\lfloor L/L_s \rfloor + 1$  segments. To investigate the influence of different *segmentation lengths*, we set  $L_s \in \{1000, 2000, 5000, 10000, 15000, 20000, 25000, 30000, 50000\}$ .

#### B. Feature Extraction

Three feature extraction methods are used in our experiment, namely term-frequency ( $tf$ ), term frequency-inverse document frequency ( $tf-idf$ ) and *ngram*. We use  $S$  to denote a system call symbol,  $T_s$  to denote the trace of a segment.  $tf(S, T_s)$  denotes the weight of symbol  $s$  in trace  $T_s$  and is computed as the occurrence frequency.

$$tf(S, T_s) = \frac{\text{count of symbol } S \text{ in } T_s}{\text{count of system calls in } T_s} \quad (3)$$

$tf-idf$  also considers how rare a system call symbol occurs in an entire trace set. We use  $T$  to represent the entire trace set and  $N_T$  to represent the total number of traces in  $T$ . The  $tf-idf$  of a symbol  $s$  is the product of term-frequency and the inverse-document frequency of that symbol.

$$tfidf(s, T_s, T) = tf(s, T_s) \cdot idf(s, T) \quad (4)$$

$$idf(s, T) = \log\left(\frac{N_T}{\text{count}(T_s \in T : s \in T_s) + 1}\right)$$

*ngram* captures the sequential information of system call traces. A trace is divided into fixed length sub-sequences, called *n-grams*, with a sliding window of length  $n$ . The feature vector is essentially the distinct *n-grams* weighted by their occurrence frequency. In our experiment, we use  $n$  equal to 3.

#### C. Kernel Functions

We use two kernel functions for the OC-SVM learning algorithm in our experiment, namely linear and Gaussian. As described in Equation 2, the kernel function computes the dot product of two feature vectors in feature space  $F$  with a function of vectors in input space  $I$ . Let  $X_i$  and  $X_j$  be two feature vectors in input space  $I$ . The linear kernel is simply the dot production of  $X_i$  and  $X_j$ .

$$K(X_i, X_j) = X_i \cdot (X_j)^T \quad (5)$$

The Gaussian kernel is computed as following:

$$K(X_i, X_j) = \exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma^2}\right) \quad (6)$$

#### D. Evaluation Metrics

To evaluate the performance of the DE, we measure four metrics, namely true positive rate (TPR), false positive rate (FPR), area under the ROC curve (AUC) and execution time.

The values of TPR and FPR are calculated as following:

$$TPR = \frac{TP}{TP + FN} \quad (7)$$

$$FPR = \frac{FP}{TP + FP} \quad (8)$$

TP (true positive) indicates the number of anomalies that are classified correctly. FN (false negative) indicates the number of anomalies that are not detected by the classifier. FP (false positive) represents the number of normal samples that are classified as anomalies.

A ROC curve is a graphical plot that illustrates the performance of a classifier with different discrimination thresholds. This curve plots TPR (y-axis) against FPR (x-axis). The OC-SVM algorithm essentially does not provide any probability score. In the experiment, we approximate the score as a function of the distance from the input data point to the decision boundary. AUC measures the total 2-dimensional area under the ROC curve. It summarizes the information of the ROC curve and measures the capability of a classifier to distinguish between positive and negative classes. The higher the AUC value, the better performance a classifier can achieve. The AUC value ranges from 0 to 1 and the classifier is perfect if  $AUC = 1$ .

## VII. PERFORMANCE OF ANOMALY DETECTION MODULE

We focused our attention to the evaluation metrics, namely TPR, FPR, AUC and execution time, of the *Anomaly Detection* module. In Figure 3 we show their values for different kernels and feature vectors as function of the *segmentation lengths*. In Table III we evaluate the same metrics for different applications and attacks.

1) *Comparison among different segmentation length*: The choice of a specific *segmentation length* is part of the DE configuration and it is important for us to determine what is the impact of its value.

As shown in Figure 3, both TPR and FPR values show a growing trend with larger *segmentation length* values for all applications and features. The TPR converges at a specific point with a value close to or equal to 1. The performance of the *Anomaly Detection* model degrades with higher FPR if the *segmentation length* exceeds that point. The performance of the DE degrades significantly with an improper *segmentation length*, particularly in the region of lower values. It is therefore vital to choose the most appropriate value.

We observe that there is a *segmentation length* which can provide optimal performance for the module. According to the experimental results of the three applications we used, the optimal *segmentation length* is 30000.

2) *Comparison among different features and kernels*: As seen in Figure 3, the feature *tfidf* performs significantly worse than the *ngram* and *tf* features for all applications. In the worst case, the FPR values can be as high as 0.78, 0.91, 0.92 and the TPR values can be as low as 0.38, 0.42, 0.15 for CouchDB, MongoDB and Image Classification respectively. This is not an acceptable performance of an IDS and there are two possible explanations. Firstly, we must observe that the *idf* factor is extracted from only the training dataset to avoid information leakage. A distortion will occur when applying it to the testing dataset, especially for the abnormal data. Secondly, our results

TABLE III: AUC, TPR, FPR values of different applications and attacks.

Application	Attack	AUC	TPR	FPR
CouchDB	Execute Arbitrary Code	0.995	1	0.067
Mongoddb	Brute Force	0.959	1	0.020
Image Classification	PGD	0.917	1	0.12
	BIM	0.949	0.972	0.12
	CW	0.929	0.988	0.12
	FAB	0.951	0.961	0.12
	MIFGSM	0.851	1	0.12
	PGDDLRL	0.857	1	0.12
	Square	0.858	1	0.12
	TPGD	0.799	0.55	0.12

indicate that the rareness of a syscall symbol across traces in the entire dataset (measured with *idf*) is not an effective indicator for real anomalies (intrusions).

*ngram* and *tf* produce nearly identical results along various *segmentation lengths* for a given kernel.

To determine the preferred feature vector we focused our attention to the execution for both of them. Figure 3d illustrates the execution time of training the model with *ngram* and *tf* features for all 3 applications with the ideal segmentation length of 30000. The execution time includes parsing the raw traces, extracting features, training and testing the model. The Image Classification with feature *tf* and linear kernel takes minimum time and we use it as the basic time unit  $T_0$  ( $T_0 = 8.2s$ ). All the execution times are represented as multiples of  $T_0$ . As shown in Figure 3d, *ngram* always takes a longer time compared to *tf* for each application and the time difference is positively related to the number of system call symbols (see Table I and Table II).

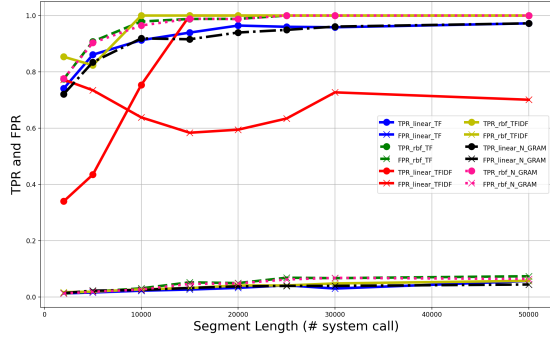
Given our results, we can conclude that the feature *tf* is the best choice since it provides almost the best detection performance with a lower workload.

The optimal kernel mainly depends on the spatial distribution of the normal (application traces) and abnormal data points (attack traces) in the input space  $I$ . Linear kernel works better if the data points are essentially linearly separable and vice versa.

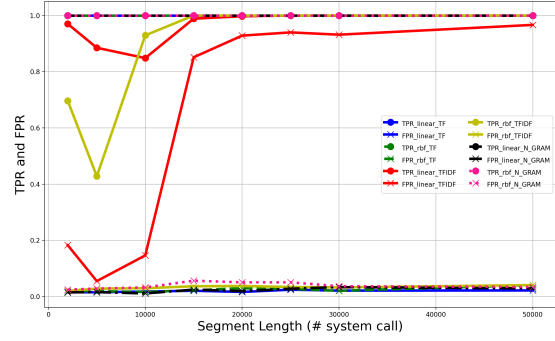
3) *Comparison among applications and attacks*: Table III summarizes the AUC, TPR and FPR values of the OC-SVM algorithm for all applications and attacks described in Section V. The OC-SVM model is trained with *tf* feature extraction method and Gaussian kernel. The *segmentation length* is set to 30000. We chose these parameters because they are the optimal choices according to our discussion in Section VII-1 and Section VII-2.

The attacks *arbitrary code execution* and *brute force* performed on dynamic applications are easier to detect. The DE is able to detect 100% of attacks at a FPR of 6.7% for *arbitrary code execution* and 2% for *brute force*. The AUC values can reach as high as 0.995 and 0.959.

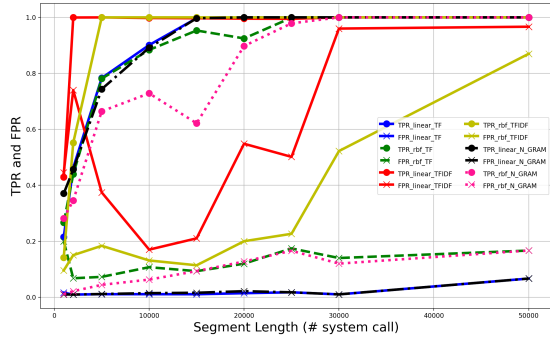
It is more difficult to detect adversarial machine learning attacks because the distinctions between normal and anomalous traces are weak. For the Image Classification application,



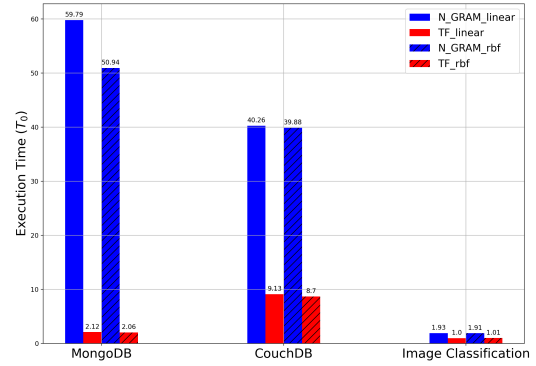
(a) CouchDB and Arbitrary Code Execution Attack



(b) MongoDB and Brute Force Attack



(c) Image Classification and Adversarial ML Attack (PCA)



(d) Execution Time

Fig. 3: The FPR and TPR values (Figure 3b, 3a, 3c) and execution time (Figure 3d) of the OC-SVM with different *segmentation lengths*, features and kernel functions. The TPR values are shown as circles and the FPR values are shown as crosses.

the FPR is relatively high (12%). The TPR rate varies with different adversarial sample generation methods. For PGD, MIFGSM, PGDDL and Square, 100% of the attacks can be detected. However, only 55% attacks of TPGD can be caught by the DE of the IDS.

### VIII. PERFORMANCE OF ANOMALY ANALYSIS MODULE

We evaluated the effectiveness of the *Anomaly Analysis* module discussed in Section IV-C. The *Anomaly Analysis* module recognizes the standalone anomalies and we measured the TPR and FPR values before and after filtering out those standalone anomalies. Figure 4 shows the metrics for different *segmentation lengths* for the MongoDB application with feature *tf* and linear kernel.

As shown in Figure 4, the TPR values are equal to 1 for all *segmentation lengths* before and after filtering. This indicates there is no additional performance loss, in terms of TPR, after filtering.

The FPR values drop significantly. The original FPR ranges from 0.013 to 0.021 with various *segmentation lengths*. After filtering, the maximal FPR (with *segmentation length* equal to 50000) is only 0.014. The values can even reach 0 when optimal *segmentation length* is chosen.

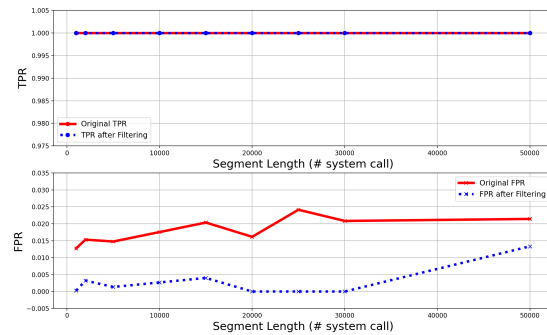


Fig. 4: The TPR and FPR values as function of the *segmentation length* before and after filtering standalone anomalies in the case of the MongoDB application with *tf* feature and linear kernel

### IX. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an intrusion detection system based on OC-SVM that monitors and analyzes system calls. For each uniquely identifiable compute object, an IDS model is

trained centrally in an authorized party and distributed to local nodes via a secure channel. An anomaly analysis model was implemented to reduce false alarms of the system. A dataset was established containing system calls for one container escalation attack, one brute force attack and a number of adversarial ML attacks.

The experimental results demonstrated that the OC-SVM algorithm can successfully detect modern attacks with satisfactory FPR, ranging from 0.02 for the brute force attack up to 0.12 for adversarial ML attacks. In addition, we observed that the system gains the optimal performance with feature extraction method *term-frequency* with TPR equal to one for large part of our attacks. Furthermore, the choice of *segmentation length* is vital. The system performance degrades significantly if the *segmentation length* is too small. For the applications we examined the optimal *segmentation length* is 30000 in terms of number of system call symbols. In addition, we observed that the optimal kernel functions are application dependent.

In the future, we aim to detect intrusions of distributed applications by monitoring multi-dimensional metrics, e.g. the network traffic among nodes of an application and CPU usage. This will allow us to achieve richer information and detect malicious behaviors more accurately. In addition, we want to extend our system to be able to apply countermeasures automatically based on the output of the DE module.

#### ACKNOWLEDGMENTS

This paper builds upon the work done within the Dutch NWO Research project ‘Data Logistics for Logistics Data’ (DL4LD, [www.dl4ld.nl](http://www.dl4ld.nl)), supported by the Dutch Top consortia for Knowledge and Innovation ‘Institute for Advanced Logistics’ (TKI Dinalog, [www.dinalog.nl](http://www.dinalog.nl)) of the Ministry of Economy and Environment in The Netherlands and the Dutch Commit-to-Data initiative (<https://commit2data.nl/>).

#### REFERENCES

- [1] L. Zhang, R. Cushing, L. Gommans, C. De Laat, and P. Grosso, “Modeling of collaboration archetypes in digital market places,” *IEEE Access*, vol. 7, pp. 102 689–102 700, 2019.
- [2] J. A. Kassem, C. De Laat, A. Taal, and P. Grosso, “The epi framework: A dynamic data sharing framework for healthcare use cases,” *IEEE Access*, vol. 8, pp. 179 909–179 920, 2020.
- [3] R. Cushing, R. Koning, L. Zhang, C. de Laat, and P. Grosso, “Auditable secure network overlays for multi-domain distributed applications,” in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 658–660.
- [4] S. Soltész, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, “Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors,” in *Proceedings of the 2Nd ACM SIGOPS/EuroSys european conference on computer systems 2007*, 2007, pp. 275–287.
- [5] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018.
- [6] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, “A study on container vulnerability exploit detection,” in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 121–127.
- [7] T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, R. A. Bridges *et al.*, “A survey of intrusion detection systems leveraging host data,” *arXiv preprint arXiv:1805.06070*, 2018.
- [8] G. Creech and J. Hu, “Generation of a new ids test dataset: Time to retire the kdd collection,” in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4487–4492.
- [9] S. A. Hofmeyr, S. Forrest, and A. Somayaji, “Intrusion detection using sequences of system calls,” *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.
- [10] W. Khreich, E. Granger, A. Miri, and R. Sabourin, “A survey of techniques for incremental learning of hmm parameters,” *Information Sciences*, vol. 197, pp. 105–130, 2012.
- [11] Y. Liao and V. R. Vemuri, “Use of k-nearest neighbor classifier for intrusion detection,” *Computers & security*, vol. 21, no. 5, pp. 439–448, 2002.
- [12] B. Subba, S. Biswas, and S. Karmakar, “Host based intrusion detection system using frequency analysis of n-gram terms,” in *TENCON 2017-2017 IEEE Region 10 Conference*. IEEE, 2017, pp. 2006–2011.
- [13] M. Xie, J. Hu, and J. Slay, “Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld,” in *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, 2014, pp. 978–982.
- [14] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, “An anomaly detection system based on variable n-gram features and one-class svm,” *Information and Software Technology*, vol. 91, pp. 186–197, 2017.
- [15] W. S. Noble, “What is a support vector machine?” *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.
- [16] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt *et al.*, “Support vector method for novelty detection,” in *NIPS*, vol. 12. Citeseer, 1999, pp. 582–588.
- [17] L. M. Manevitz and M. Yousef, “One-class svms for document classification,” *Journal of machine Learning research*, vol. 2, no. Dec, pp. 139–154, 2001.
- [18] L. Zhang., R. Cushing., R. Koning., C. de Laat., and P. Grosso., “Profiling and discriminating of containerized ml applications in digital data marketplaces (ddm),” in *Proceedings of the 7th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP, INSTICC*. SciTePress, 2021, pp. 508–515.
- [19] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” *arXiv preprint arXiv:1611.01236*, 2016.
- [20] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning,” *Pattern Recognition*, vol. 84, pp. 317–331, 2018.
- [21] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, “Towards deep learning models resistant to adversarial attacks,” *arXiv preprint arXiv:1706.06083*, 2017.
- [22] Y. Bengio and Y. Grandvalet, “No unbiased estimator of the variance of k-fold cross-validation,” *Journal of machine learning research*, vol. 5, no. Sep, pp. 1089–1105, 2004.