A clustered learning framework for host based intrusion detection in container environment

Jingfei Shen[*‡], Fanping Zeng[†§], Weikang Zhang[*‡], Yufan Tao[*‡], Shengkun Tao[*‡]
[*]*School of Computer Science and Technology, University of Science and Technology of China, Anhui, China*
[†]*Key Laboratory of Wireless-Optical Communications, University of Science and Technology of China*
[‡]{ericjeff, buttman, tyf1999, nonestack}@mail.ustc.edu.cn
[§]billzeng@ustc.edu.cn

2022 IEEE International Conference on Communications Workshops (ICC Workshops) | 978-1-6654-2671-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCWORKSHOPS53468.2022.9814620

***Abstract*—Container technology has been widely deployed in edge computing environments. Using the OS-level resource isolation and management, it can achieve incomparable high efficiency in contrast to the traditional virtual machine approach. However, recent studies have shown that the container environment is vulnerable to various security attacks. Furthermore, the highly customizable and dynamic change nature of the container exacerbated its vulnerability. In this paper, we propose a new anomaly detection framework combining cluster algorithm to improve anomaly detection efficiency in the edge computing environment that contains a large number of containers. It utilizes cluster algorithm to automatically identify containers that running the same application, and builds an anomaly detection model for each category separately. We investigated 8 real-world vulnerabilities from several frequently used applications and evaluated our framework on them. Experiment results show that our proposed framework can effectively identify containers built on the same application image without any manual labeling, reduce the FPR of anomaly detection from 0.61% to 0.09%, and increase TPR from 90.3% to 96.2% compared to the traditional method.**

***Index Terms*—Container Security, Edge Computing Environment, Intrusion Detection, Cluster, Machine Learning**

I. Introduction

Container technology has been widely adopted in today's edge computing environment due to its scalability, high efficiency, and low overhead of isolation. Container technology is an operating system (OS) level virtualization technology that has multiple building blocks inside the Linux kernel, including resource isolation, control techniques (e.g., namespace and cgroup) and security mechanisms (e.g., Capabilities, SELinux [1], AppArmor [2], and Seccomp [3]). By avoiding the overhead of additional abstraction layers, containers can achieve near-native performance and outperform virtual machine-based systems in almost all aspects [4]. In addition, the advent of container management and orchestration systems, such as Docker and Kubernetes, has greatly changed the ecosystem of building, shipping, and deploying distributed applications in the edge computing environment.

Present research on container security includes static analysis of container images and anomaly detection of a running container. The former analyzes the security problems by detecting binary files, scripts, libraries, and other files in the image. The latter analyzes the resource usage and log data of the container to detect the anomaly behavior of the container. In this paper, we focus on the latter approach, studying the detection method for anomaly behavior of containers caused by security attacks, i.e., anomaly-based intrusion detection. An intrusion detection system (IDS) is a device or software application that monitors a network or system for malicious activity or policy violations [5]. Methods commonly used for intrusion detection include signature-based and anomaly-based. The signature-based method detects anomalies through pattern matching, such as byte sequences of network traffic and known instruction sequences of malware. It can effectively detect malicious attacks with known patterns, but cannot identify unknown malicious attacks [6]. The anomaly-based method is mainly used to detect unknown attacks. It builds a model of trusted normal behaviors through machine learning approaches and uses the built model detecting anomaly.

Container technology has brought new challenges to intrusion detection. Due to the lightweight and efficient characteristics, a host machine often contains multiple container instances, intrusion detection system of the host machine, therefore, cannot effectively detect anomaly behaviors of each container. In addition, because of the wide variety and highly customizable nature of the container, it's always difficult to generalize the detection model among different containers, while building a model for every container will incur a large performance overhead. Therefore, new methods are urgently needed to improve anomaly detection of containers.

In this paper, we proposed a framework to improve container anomaly behavior detection using clustering. The proposed framework utilizes cluster algorithm to identify containers running the same application without any manual labeling, build a model for every application and perform anomaly detection for each of them. The main contributions of this work are as follows:

- A new anomaly detection framework combined with unsupervised classification (i.e. cluster) is proposed to improve intrusion detection efficiency in the container environment.
- We present efficient container classification and anomaly detection by integrating cluster algorithm with lightweight machine learning methods.
- This work shows how system calls can be used to inspect the behavior of a Container and to perform classification and anomaly detection.

This paper is organized as follows: Section II presents related work, which investigates anomaly detection based on system calls. Section III presents our proposed approach of anomaly detection, which is then evaluated in Section IV. The paper concludes in Section V.

II. Related Work

Container security. Container security issues have attracted more and more attention in recent years. Shu et al. [8] investigated the vulnerability of images in container hosting platforms. They analyzed 356,218 images and found that both official and community images contain an average of 180 vulnerabilities, and vulnerabilities in images are passed from the base image to the derived image with dependency. Combe et al. [9] analyzed the possible information leakage channels between containers and host and expounded the potential security risks, such as private data inference and same host verification, etc. Gao et al. [10] explained the source of main attacks of Docker container, compared the security features of the container and Linux kernel, and mentioned the security problem of using container technology to directly communicate with and attack the host kernel. Apart from the vulnerability, there also is much research on container safety protection. Yuqiong et al. [11] implemented the Security namespace in the Linux kernel, which allows containers to customize security policies independently and apply them to specific processes. Sergei et al. [12] proposed SCONE, a safeguard against SGX Trusted Execution Technology using Intel processors. This method can encrypt/decrypt the I/O data at a low cost, guarantee the data of the container not be used by the attacker, and enhance the container security.

System call-based anomaly detection. The basic idea of anomaly detection is to create a model for trusted operations through machine learning method, then use the model to detect abnormal behavior. This method mostly uses system call sequences of the process as data. Kang et al. [13] proposed a representation method called bag of system calls(BoSC) for anomaly detection. This method divides the system call sequence into segments, counts the frequency of different system calls in each segment, then uses the statistical sequence for anomaly detection. They took detection as a classification problem and used machine learning to detect anomalies. The results showed that the accuracy and false positive rate were significantly improved compared with the previous methods.

Most of the anomaly detection based on system call only uses the name of the system call as features, but for deliberately constructed attack methods such as mimicry attack, it often cannot find abnormal behavior in time. Parampalli et al. [14] proposed an interactive mimicry attack method, in which the attacker could constantly modify the data and embedded code in the malicious script based on the results of the attack, thereby bypassing the intrusion detection system. In this regard, Larson et al. [15] proposed that the use of additional information in the system call, such as the parameters of the system call, the return value, the user ID of the caller, can effectively improve the success rate of anomaly detection.

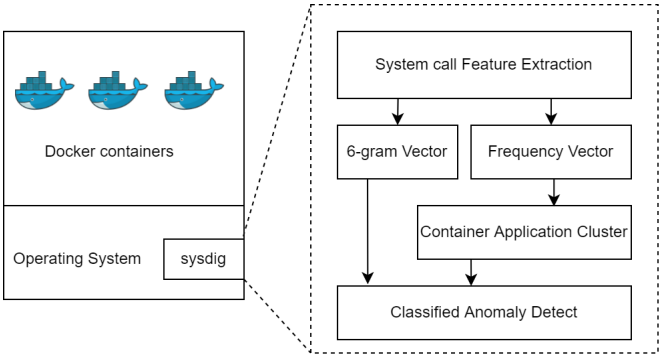

Fig. 1. Framework overview

Container anomaly detection. For a started container, essentially it is a set of processes within the host system. Thus the anomaly detection based on system call is also applicable to containers. Abed et al. [17] proposed to apply the bag of system calls to anomaly detection of containers. They store normal system calls in a behavior database. When detection, if the current system call sequence does not appear in the database, it is marked as an anomaly. If the marked anomaly reaches a certain threshold, the behavior is considered to be an anomaly. Karn et al. [18] proposed a detection method to solve the problem that containers are used to carry out cryptocurrency operations in high-performance computing. their experiment results show that the accuracy of the decision tree is 97.1%, which is significantly higher than other machine learning methods. Current detection methods for containers mostly focus on specific fields and can not apply to the edge computing environment with various kinds of and a huge amount of containers. Therefore, an anomaly detection framework that can be generalized to different container applications is needed.

III. Design of Proposed Framework

A. Framework Overview

Our proposed framework consists of three main components: system call data collector, container classifier, and anomaly detector. Fig. 1 shows an overview of the proposed framework. An open-source, out-of-the-box tool, sysdig [7], is used to obtain system call data generated by containers to achieve non-intrusion, low-cost anomaly detection. For container classification, the DBSCAN [19] cluster algorithm is used to classify containers in an unsupervised way. During detection, the framework builds a RandomForest [20] classifier for each application category to detect anomalies in observed system calls.

B. System Call Data Collector

We monitor the system call data generated by the container using the sysdig tool, which provides native support for Linux containers including Docker, LXC, etc. System call contains a lot of information such as system call name, parameters, and return value. During the experiment, we found the name

TABLE I
FREQUENCY VECTOR OF HTTPD

	system call frequency				
Timestamp	**poll**	**mmap**	**read**	**close**	**writev**
1618900524618	0.0151	0.0640	0.2213	0.0987	0.0764
1618900524718	0.0156	0.0635	0.2309	0.1013	0.0804
1618900524818	0.0173	0.0643	0.2284	0.0960	0.0779
1618900524918	0.0272	0.0549	0.2546	0.0926	0.0883
1618900524019	0.0122	0.0721	0.2099	0.1026	0.0708
1618900524128	0.0274	0.0602	0.2250	0.0904	0.0848
1618900524229	0.0230	0.0588	0.2561	0.0964	0.0793
1618900524329	0.0285	0.0560	0.2445	0.0859	0.0896
1618900524430	0.0301	0.0644	0.2306	0.0964	0.0692

TABLE II
FREQUENCY VECTOR OF MYSQL

	system call frequency				
Timestamp	**poll**	**mmap**	**read**	**close**	**writev**
1618900524618	0.0068	0.0000	0.2666	0.0034	0.0000
1618900524718	0.0057	0.0000	0.2859	0.0029	0.0000
1618900524818	0.0010	0.0000	0.3190	0.0000	0.0000
1618900524918	0.0068	0.0000	0.2754	0.0045	0.0000
1618900524019	0.0022	0.0000	0.2968	0.0022	0.0000
1618900524128	0.0059	0.0000	0.2831	0.0022	0.0000
1618900524229	0.0033	0.0000	0.3012	0.0022	0.0000
1618900524329	0.0019	0.0000	0.3151	0.0010	0.0000
1618900524430	0.0020	0.0000	0.3104	0.0007	0.0000

of system call is sufficient for container classification and anomaly detection. Thus, we specify necessary parameters for sysdig to filter out other information and get the system call name sequence of a running container.

The collected system call name sequence is essentially a type of time-series data. Linux system contains approximately 420 system calls [21], which can be regarded as discrete system events. For clustering, the collected name sequence is continuously sampled at a specific time interval and processed into many frequency vectors similar to the One-Hot encoding method [22]. The dimension of the frequency vector is the number of different system calls (i.e. 420). Table I and Table II show two examples of frequency vectors generated by an *Httpd* container and a *MySQL* container. Each timestamp corresponds to a frequency vector $V = [f_1, f_2, ..., f_k]$, where f_k represents the frequency ratio that the corresponding system call was executed by the container within the sampling interval.

C. Container Classification

The purpose of classification is to identify containers built by the same application image from the perspective of their behavior. Due to its highly customizable characteristics, containers in the edge computing environment often have no detailed application category information, and the large amount of them makes it difficult to label manually. Moreover, behaviors of different applications diverge greatly from each other, causing the anomaly detection model that works well for one application not suitable for the other. Table II shows an example of a partial frequency vector generated by the MySQL container. In contrast to Table I, it can be seen that system calls such as *mmap*, *writev*, and *fstat*, which were executed multiple times by the Httpd container, were never executed in the MySQL container. Therefore, unsupervised classification (i.e. cluster) is used to achieve effective classification on unlabeled system call data.

In terms of the clustering method, DBSCAN [19] is chosen for clustering containers. This method conducts unsupervised classification according to density. Compared with K-means [23], it could find clusters of any shape instead of being limited to a convex shape [24]. During clustering, every sampled frequency vector generated by a container will participate in the clustering process, and their corresponding clustering results are obtained according to the DBSCAN algorithm. After that, the category with the largest number of clustering results of all frequency vectors will be regarded as the category of the container.

D. Anomaly Detection

As for anomaly detection, the collected system call data of a container is first mapped into word indexes, then sampled into n-gram terms, which will finally be fed into a RandomForest classifier to detect anomalies in the data. N-gram model is a commonly used method in Natural language Processing, it is a contiguous sequence of n items from a given sample of text or speech. There are other standard approaches like the principal component analysis (PCA) used in [25] to extract features from the system call sequence. PCA is known to be computationally intensive and therefore unsuitable for real-time detection. We sample system calls into contiguous 6-gram terms. As an example, consider the system call sequence $\{1, 4, 2, 6, 12, 34, 12, ...\}$, with the 6-gram method, it will be sampled into frames such that $frame_1 = \{1, 4, 2, 6, 12, 34\}$, $frame_2 = \{4, 2, 6, 12, 34, 12\}$, $frame_3 = \{2, 6, 12, 34, 12, ...\}$ and so on. The n-gram terms are split into training and validation set as the standard machine learning convention (ratio of 70:30).

As described above, our proposed framework build an anomaly detection model for every application category according to the cluster results. For per category detection, a RandomForest classifier is used to detect anomalies. RandomForest is an ensemble learning method for classification, regression, and other tasks that operates by constructing a multitude of decision trees at training time to improve accuracy while combating over-fitting. Fig. 2 shows the simplified diagram of a random decision forest.

Each decision tree makes local optimal splitting decisions among a random subset of the features when splitting nodes. As a result, the trained decision trees are usually quite different from one another. The output of the random forest classifier then uses the majority voting result among individual decision trees. Many other machine learning methods can be used to perform anomaly detection, such as the K-NearestNeighbor algorithm (KNN) [26], Support Vector Machine(SVC) [27] and Neural Network. However, by evaluating them on the ADFA-LD dataset [28], we found that RandomForest outperforms other methods and is therefore selected as anomaly detectors.

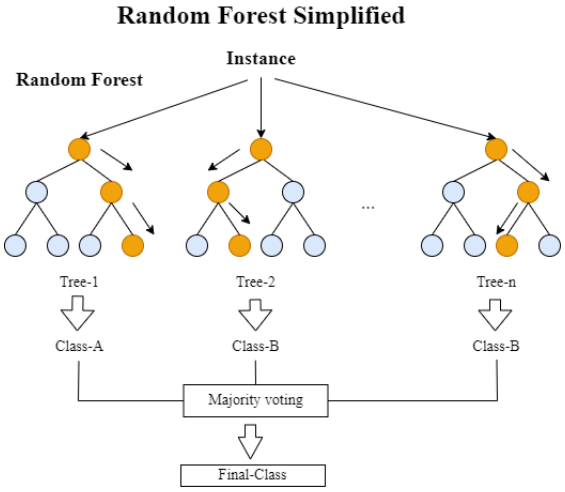

Fig. 2. Diagram of a random forest

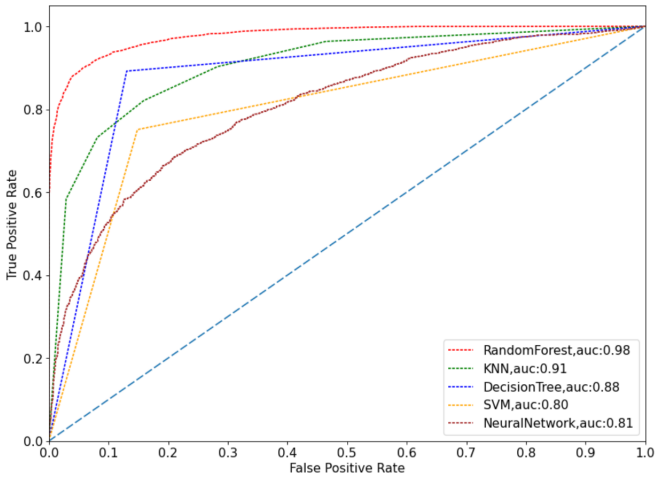

Fig. 3. ROC curve of different algorithm on ADFA-LD dataset

Fig. 3 shows the ROC-AUC curve of the evaluation results of different algorithms. ROC curve is a graph that shows the performance of a classification model at all possible thresholds and AUC is the entire area beneath this ROC curve. From the curve, we can see that RandomForest achieves the best AUC score of 0.98, which is far better than other algorithms.

IV. Experiment Setup and Evaluation

In this section, we present our experiment environment setup and result analysis. We implement our detection framework and evaluate it on a Dell PowerEdge T440 server with 40 2.20GHz CPUs and 64GB memory running Ubuntu 20.04.1 LTS system.

A. Experiment Setup

1) Normal and anomaly data: We selected 8 vulnerabilities of 4 open-source software from the Common Vulnerability Exposures(CVE). These vulnerabilities cover security issues related to SQL injection, remote command injection, arbitrary file write, etc. Table III shows information about each vulnerability. Vulhub [29] was used to set up our experimental environment. It provides a lightweight vulnerable environment based on Docker containers.

TABLE III
List of Vulnerabilities

Application	Vulnerabilitiy	Type
Django	CVE-2017-12794	XSS Attack
Django	CVE-2019-14234	SQL Injection
Django	CVE-2021-35042	SQL Injection
Httpd	CVE-2017-15715	Security Policy Bypass
Httpd	bash-shellshock	Remote Command Injection
MySQL	CVE-2012-2122	Identity Bypass
Tomcat	CVE-2017-12615	Arbitrary File Write
Tomcat	CVE-2020-1938	Arbitrary File Read

System calls generated by idle applications are usually system calls related to the user-kernel space switch, which are very homogeneous even for different applications. Therefore, we use JMeter [30] to simulate the application workload. It can be used as a unit-test tool for JDBC database connections, web services, HTTP, etc. In this work, we use JMeter's HTTP sampler to send requests to an application that provides web service, and its JDBC sampler to send database read/write requests to a database application. As for anomaly data, we execute the vulnerability script while the container is running and record the timestamp and names of system calls. We use sysdig to extract system call data of containers. Specifically, we first simulate the workload for 30 seconds, collecting names of system calls and their corresponding timestamps, which are sampled into frequency vectors for clustering. After that, we utilize the vulnerability exploit script to attack the container, getting system calls containing anomalies. These data are then sampled into 6-grams together with the normal ones and used to evaluate the efficiency of the anomaly detector.

2) Clustering: Frequency vectors are obtained by sampling system calls from containers at a 10ms interval. The value in the frequency vector reflects how often the corresponding system call is invocated at the interval. Through observation, we find that applications under normal workload generate more than 200 system calls over a 10ms interval, thus, Frequency vectors containing less than 20 system calls are discarded. After sampling all system calls generated, we use the scikit-learn implementation of the DBSCAN algorithm to cluster the frequency vectors of all containers. The Euclidean distance is used as the distance metric between two frequency vectors of the DBSCAN algorithm. In addition, two parameters eps and $minPts$ are set to 0.2 and 5 respectively to achieve reasonable clustering results. Finally, the category of a container is the label that has the maximum number of system calls in the cluster results.

3) RandomForest Classifier: The scikit-learn implementation of random forest is used to perform anomaly detection. Internally, RandomForest creates multiple Decision Trees to classify the same observation. The created decision tree relies

on a set of rules derived from the training process to partition data into groups that are as homogeneous as possible. In our work, a RandomForest with 100 estimators is built to evaluate the detection efficiency.

4) Alternative detection approaches: We evaluate our proposed approach against two commonly used detection methods in the edge computing environment.

One detector for all containers. This method aggregates system call data from all containers for anomaly detection without distinguishing the application category of the container. Due to the large variety of containers in the edge computing environment and the differences in system call sequences generated by different applications, this method may be less effective for detection.

One detector for each container. This method builds a model for each container to detect anomalies and can achieve considerable results on a single container. For a large number of containers, however, this approach incurs a performance cost that cannot be ignored, making it hard to carry out.

5) Evaluation metrics: True Positive Rate (TPR, also known as sensitivity) and False Positive Rate (FPR, also known as false alarm ratio) are used to compare the detection results with different approaches. The calculation for these metrics are given by the following equations where TP denotes the number of samples that the detector correctly identifies while FP denotes the number of samples that the detector falsely identifies and TN denotes the number of the samples the detector correctly rejects while FN denotes the number of the samples the detector falsely rejects.

$$TPR = \frac{TP}{TP + FN} \tag{1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2}$$

B. Result Analysis

1) Container Cluster: The purpose of clustering is to effectively identify containers built from the same application image. We compare the clustering results of the DBSCAN algorithm used by the proposed framework with the classical algorithm KMeans. Fig. 4 and Fig. 5 show the cluster results of DBSCAN and KMeans. It presents the ratio of system calls that are labeled as different categories in the clustering results for each container. It can be seen that for DBSCAN, most system calls for the same application are labeled as the same class, indicating that they can effectively identify containers running the same application. For example, three Django containers labeled as $label\_0$, two containers running the Apache Tomcat application are labeled as $label\_3$. For KMeans, however, it could not correctly identify the same container and mistakenly label containers of different applications as the same category, e.g. two Httpd containers labeled as different categories.

C. Clustered Detection

Fig. 6 shows the FPR of the three approaches across all application categories. It can be seen that the proposed approach, i.e. one detector for each application category, achieves the smallest FPR of 0.10% on MySQL and 0.016% on Tomcat. While on Django and Httpd applications, the proposed approach achieves a similar lower FPR of 0.09% compared with the one detector for each container approach. The one detector for all containers approach, however, achieves the highest FPR of 0.6% due to its incapability of modeling a large number of system calls from various applications. Fig. 7 shows the TPR of the three approaches across all application categories. It can be seen that the proposed framework increase the TPR to 95.9% from 91.2% on the Apache Tomcat application Compared to the one detector for all approaches, while on the other three application categories, it achieves almost the same TPR of 97% as the other two methods.

From the experimental results, it can be found that, due to the limited fitting ability of a single model, the one detector for all containers approach performs the worst, the one detector for each container achieves the overall best results but needs to establish a detection model for each container. The framework we proposed can achieve comparable or even better detection efficiency than the one detector for each container method, while only needs to build a detection model for each application category.

V. Conclusion

In this paper, we proposed a framework to improve anomaly detection in container environments by clustering containers into different application categories and building a detection model for each category. We show that with the DBSCAN algorithm, system call data can be used to classify containers without any manual labeling, and effectively identify containers running the same application. By building an anomaly detection model for each application category, our proposed framework can reduce the false positive rate from 0.6% to 0.09% compared to the one detector for all approaches, and achieve better detection results compared to traditional anomaly detection approaches.

Acknowledgment

This work is supported partly by the National Key R&D Program of China 2018YFB2100300, 2018YFB0803400, and the National Natural Science Foundation of China (NSFC) under grant 61772487.

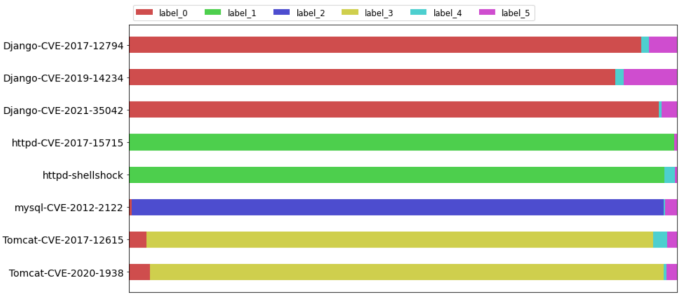

Fig. 4. Cluster results of DBSCAN

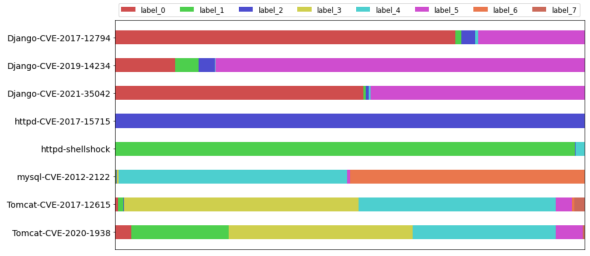

Fig. 5. Cluster result of kmeans

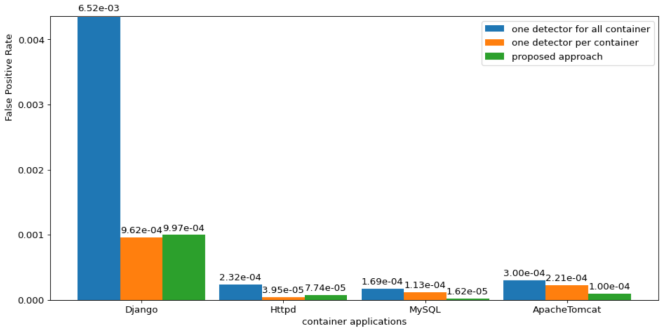

Fig. 6. Detection FPR among different approaches

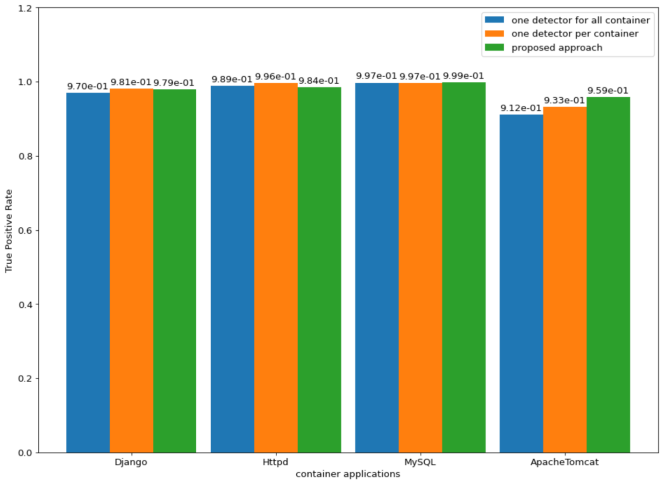

Fig. 7. Detection TPR among different approaches

REFERENCES

[1] 2021. Security-Enhanced Linux. https://en.wikipedia.org/wiki/Security-Enhanced_Linux.

[2] 2021. AppArmor. https://en.wikipedia.org/wiki/AppArmor.

[3] 2021. Seccomp. https://en.wikipedia.org/wiki/Seccomp.

[4] Containers Not Virtual Machines Are the Future Cloud. http://www.linuxjournal.com/content/containers.

[5] 2021. Intrusion detection system. https://en.wikipedia.org/wiki/Intrusion_detection_system.

[6] Douligeris, Christos; Serpanos, Dimitrios N. (2007-02-09). Network Security: Current Status and Future Directions. John Wiley & Sons. ISBN 9780470099735

[7] 2021. Secure DevOps for Containers, Kubernetes, and Cloud. https://docs.sysdig.com.

[8] Shu R, Gu X, Enck W. A study of security vulnerabilities on docker hub[C]//Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. 2017: 269-280.

[9] Combe T, Martin A, Di Pietro R. To docker or not to docker: A security perspective[J]. IEEE Cloud Computing, 2016, 3(5): 54-62.

[10] Gao X, Steenkamer B, Gu Z, et al. A study on the security implications of information leakages in container clouds[J]. IEEE Transactions on Dependable and Secure Computing, 2018.

[11] Sun Y, Safford D, Zohar M, et al. Security namespace: making linux security frameworks available to containers[C]//27th USENIX Security Symposium (USENIX Security 18). 2018: 1423-1439.

[12] Arnautov S, Trach B, Gregor F, et al. SCONE: Secure linux containers with intel SGX[C]//12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). 2016: 689-703.

[13] Kang D K, Fuller D, Honavar V. Learning classifiers for misuse and anomaly detection using a bag of system calls representation[C]//Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop. IEEE, 2005: 118-125.

[14] Parampalli C, Sekar R, Johnson R. A practical mimicry attack against powerful system-call monitors[C]//Proceedings of the 2008 ACM symposium on Information, computer and communications security. 2008: 156-167.

[15] Larson U E, Nilsson D K, Jonsson E, et al. Using system call information to reveal hidden attack manifestations[C]//2009 Proceedings of the 1st International Workshop on Security and Communication Networks. IEEE, 2009: 1-8.

[16] Sakurada M, Yairi T. Anomaly detection using autoencoders with nonlinear dimensionality reduction[C]//Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis. 2014: 4-11.

[17] Abed A S, Clancy T C, Levy D S. Applying bag of system calls for anomalous behavior detection of applications in linux containers[C]//2015 IEEE Globecom Workshops (GC Wkshps). IEEE, 2015: 1-5.

[18] Karn R R, Kudva P, Huang H, et al. Cryptomining Detection in Container Clouds Using System Calls and Explainable Machine Learning[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(3): 674-691.

[19] Ester, Martin; Kriegel, Hans-Peter; Sander, Jörg; Xu, Xiaowei (1996). Simoudis, Evangelos; Han, Jiawei; Fayyad, Usama M. (eds.). A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96). AAAI Press. pp. 226–231.

[20] Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001.

[21] 2021. syscalls(2) - Linux manual page. https://man7.org/linux/man-pages/man2/syscalls.2.html.

[22] Harris, David and Harris, Sarah (2012-08-07). Digital design and computer architecture (2nd ed.). San Francisco, Calif.: Morgan Kaufmann. p. 129. ISBN 978-0-12-394424-5

[23] S. Lloyd, ”Least squares quantization in PCM,” in IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129-137, March 1982, doi: 10.1109/TIT.1982.1056489.

[24] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[25] J. Zhang, K. Zhang, Z. Qin, H. Yin, and Q. Wu, “Sensitive system calls based packed malware variants detection using principal component initialized multilayers neural networks,” Cybersecurity, vol. 1, no. 1, 2018, Art. no. 10.

[26] 2021. k-nearest Neighbors algorithm. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm.

[27] Platt, John (1999). “Probabilistic outputs for support vector machines and comparison to regularizedlikelihood methods.”

[28] G. Creech. Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks, 2014.

[29] 2021. Pre-Built Vulnerable Environments Based on Docker-Compose. https://github.com/vulhub/vulhub.

[30] 2021. Apache JMeter. https://jmeter.apache.org/.