AI-DRIVEN CONTAINER SECURITY APPROACHES FOR 5G AND BEYOND: A SURVEY

Ilter Taha Aktolga[1], Elif Sena Kuru[1], Yigit Sever[1], Pelin Angin[1]
[1]Middle East Technical Univerity, Turkey

NOTE: Corresponding author: Yigit Sever, yigit@ceng.metu.edu.tr

Abstract – *The rising use of microservice-based software deployment on the cloud leverages containerized software extensively. The security of applications running inside containers, as well as the container environment itself, are critical for infrastructure in cloud settings and 5G. To address security concerns, research efforts have been focused on container security with subfields such as intrusion detection, malware detection and container placement strategies. These security efforts are roughly divided into two categories: rule-based approaches and machine learning that can respond to novel threats. In this study, we survey the container security literature focusing on approaches that leverage machine learning to address security challenges.*

Keywords – Anomaly detection, container, intrusion detection, machine learning

1. INTRODUCTION

Containers are lightweight and portable abstractions that contain the binary of an application, as well as the necessary and sufficient minimal dependencies to run them. Using containers to deploy software on the cloud has replaced bare metal installations as the industry standard [1] due to microservice-based architecture's demand for scalable and lightweight computation environments. Companies such as Amazon, Netflix, Spotify and Twitter use microservices architecture in their products [2], which is becoming increasingly commonplace in many enterprise systems. Compared to virtual machines, containers are faster to initialize and more lightweight since they do not need an extra virtualization layer to operate [3].

The widespread adoption of 5G networks has led to an increase in the use of container technologies to support the deployment and management of applications. Containers are a straightforward answer for running the services required by 5G, they are portable and lean in terms of size requirements and lightweight in terms of preparation and startup times. The use of containers in 5G networks can provide a number of benefits for Virtual Network Functions (VNFs), including improved scalability, flexibility, and efficiency.

The portability and flexibility of containers enable them to be deployed on demand, making it easier to manage the lifecycle of VNFs and to adapt to changing network conditions. Additionally, the use of container orchestration platforms such as Kubernetes allow for automated scaling and management of containerized VNFs, further improving the agility and scalability of 5G networks. This makes it easy to deploy and run VNFs on any infrastructure, and to scale them up or down as needed. Furthering the security and reliability of containers will, in turn, allow their rapid adoption in 5G VNFs [4].

With the increasing adoption of container technology, there is a growing concern about the security of containerized applications and networks. Containers are found to be less secure than virtual machines which is a detriment to their adoption [5]. The use of containers can introduce new vulnerabilities and risks that need to be addressed to ensure the security and integrity of 5G networks.

In the context of 5G networks, security is a paramount concern due to the critical nature of the services and applications being deployed. The integration of container technology in 5G networks introduces unique security challenges that need to be addressed to ensure the integrity, confidentiality, and availability of network resources and sensitive data. The lightweight and portable nature of containers, combined with their ability to dynamically scale and adapt, makes them an attractive target for adversaries seeking to exploit vulnerabilities and launch attacks. Therefore, it is essential to develop robust security measures and solutions specifically tailored for containerized environments in 5G networks. This study aims to address this need by focusing on the application of machine learning approaches to enhance container security in the context of 5G networks. By leveraging machine learning techniques, we aim to improve anomaly detection, intrusion detection, malware detection, and other security aspects within containerized 5G environments. The findings of this research contribute directly to the overarching goal of ensuring the security and trustworthiness of 5G networks, enabling the safe and reliable operation of critical services and applications.

Machine Learning (ML) techniques for container security have been investigated in many studies. Nassif et al. [6] conducted a systematic review that analyzes

machine learning models for anomaly detection. They reviewed ML models from four perspectives: the application of anomaly detection, the type of ML technique, the ML model accuracy, and the anomaly detection technique, i.e. whether they are supervised, semi-supervised or unsupervised. A review conducted by Mohan et al. [7] focused on the applications of various ML and deep learning methods in the implementation of defensive deception. They summarized the classification of several deception categories, new machine learning and deep learning techniques in defensive deception, including the models, common datasets, key contributions, and limitations. Zhong et al. [8] introduced a taxonomy of the most common machine learning algorithms used in the field of container orchestration. The authors presented ML-based container orchestration approaches, classified the orchestration methods, and demonstrated the evaluation of ML-based approaches used in recent years. Also, the authors discussed machine learning approaches for anomaly detection. Another survey conducted by Wong et al. [9] provides a systematic review of containers, covering vulnerabilities, threats, and existing mitigation strategies, to provide information on the landscape of containers. The authors also discuss some machine learning methods, and the papers utilizing ML techniques to improve container security.

This survey distinguishes itself by integrating state-of-the-art artificial intelligence methodologies, including artificial neural networks, machine learning, and deep learning techniques. It provides an extensive examination of a diverse array of detection models, encompassing supervised, semi-supervised, and unsupervised approaches. The survey employs meticulously curated datasets for the purpose of training and evaluation. Focusing specifically on fortifying containerized environments, it explores critical aspects such as intrusion detection, malware detection, attack detection, anomaly detection, and inter-container security. By offering invaluable insights into the latest advancements and persistent challenges in container security, this survey serves as an all-encompassing resource for researchers and practitioners seeking to enhance their container security measures.

2. PRELIMINARIES

In this survey, we base our discussion on the most prevalent container architecture for academia as well as the commercial space, Linux containers. Containers leverage two important Linux kernel features: control groups (cgroups) and namespaces. A namespace is a layer of abstraction that covers the processes inside that namespace. Wrapped processes get a private and isolated view of system resources. The processes inside the namespace are also isolated from the changes that happen to global resources, allowing developers to prepare environments for binaries to run with defaults that the binaries expect and not disturb execution flow. There are different types of namespaces that correspond to different constrained

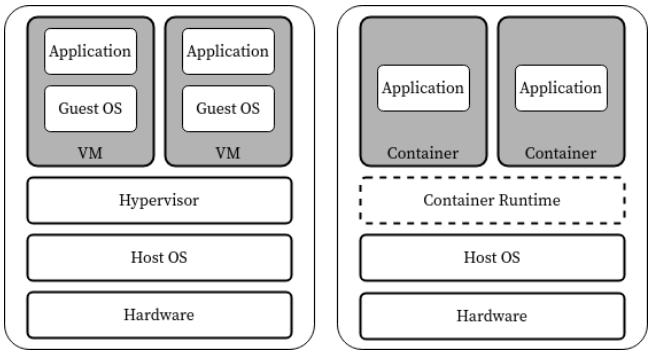

Fig. 1 – An overview of virtual machines and containers

views into system resources. There are a total of eight different namespace types which constrain the view of either the cgroup root directory, message queues, network devices, mount points, process ID space, clocks, user and group IDs and hostnames.

As namespaces wrap processes with an isolated view of system resources, the level of allocation of said resources are controlled through cgroups. cgroups are another Linux kernel feature which limits and monitors the resource usage of processes. When a process is put into a cgroup hierarchy, its access to system memory, CPU, priority of network communication, network bandwidth it can use etc. are controlled. Control groups are used to allocate system resources fairly between different containers in the same host system.

All in all, containers are Linux processes that are constrained and isolated through aforementioned kernel features. Through constraining and isolating the process or a bundle of related processes with files relevant to their operation, we get lightweight containers that can be packed with their dependencies. Since setting up containers with cgroups and namespaces can get cumbersome, there are container management frameworks and container runtimes to assume these tasks. Well-known examples of these container technologies are Docker, Podman, Linux Containers (LXC), RKT and CRI-O. A full-fledged cluster needs additional management and tooling as well. These include service discovery within the cluster, container orchestration and networking among others [10]. Docker is favored as the main container technology [11] with Docker Hub, a public container library, playing a major role in its popularity.

Although the Linux kernel provides ease of use for the isolation framework we have discussed, it comes with a drawback. Since all containers share the single kernel running on the host and there is no need for a separate hypervisor layer as in virtual machines, the isolation guarantees for containers are brittle. Vulnerabilities and mismanaged containers can cause this isolation to be broken. The result has been titled “container escape” [12].

Virtual machines are slower to start up and get running compared to containers. One canonical question can arise at this point of the discussion: why do we use virtual machines if containers are more lightweight? The security of containerized applications have been challenging

researchers while the stronger isolation offered by virtual machines are better. Furthermore, the kernel features that allow containers as we know them today have been matured much later than the framework to support virtual machines.

2.1 Intrusion Detection Systems (IDS)

An intrusion in the context of computer security is attempted or successful access to confidential data or resources by unauthorized parties. Network engineers use Intrusion Detection Systems (IDSs) which monitor a system and its resources to detect and report intrusions [13]. Monitoring system resources involves either placing sensors on host systems that analyze machine behavior or placing sensors on the network to monitor traffic. IDSs are categorized according to these sensors: Host-based IDS (HIDS) and Network-based IDS (NIDS), respectively. Machine behavior that HIDS leverage can involve CPU, RAM and disk usage and network traffic that NIDS monitor involves individual packets that flow through the network and analytics that are derived from them [14].

Another categorization we can apply to IDSs is whether they detect anomalous behavior by comparing against a set of predefined malicious behavior signatures or learning benign and malicious behavior to detect abnormal behavior. The former is named signature-based IDS and the latter is named anomaly-based IDS. Since developing signature-based IDS involves collecting and collating a large dataset which is not readily transferable from system to system [15], studies often focus on anomaly-based IDS research.

2.2 System calls

System calls are an interface between the hardware and the user space processes. Processes interact with the kernel and request privileged actions, such as interacting with hardware resources or performing network operations. These actions are restricted to certain processes and the kernel implements security policies to determine which processes can make certain system calls. Since system calls are always present whenever a process performs a worthwhile action, it offers a valuable source of information. Hence, system call monitoring is a common technique for detecting suspicious behavior in compromised applications because malicious code has to use system calls to perform malicious operations. Tools like strace and ftrace are used to show the sequence of system calls made by a particular command or process [16]. Monitoring system calls can help identify and mitigate problems caused by compromised applications.

Bag of System Calls (BoSC) [17] is a method for using system call data in machine learning applications. The method involves creating a frequency list $S = \{s_1, s_2, \dots, s_n$ where s_i is the number of times the system call during that time window is observed [18]. BoSC representation has seen frequent use in container intrusion detection literature [19, 20] often paired with the Sysdig [1] tool [21, 22] to directly stream system calls from running containers with a low overhead.

Frequency lists are not the sole method for using system call traces in machine learning applications. For instance, Srinivasan et al. [16] used sequences of system calls with preserved order to create n-grams with Maximum Likelihood Estimator for anomaly detection in containers. Karn et al. [23] used n-gram representation as well during detecting malicious processes inside containers. Iacovazzi and Raza [24], on the other hand, represented system calls in a sequence in a graph representation to preserve dependencies between system calls. In a similar vein, Chen et al. [25] represented remote procedure calls with a graph to monitor microservice behavior.

3. CYBERATTACKS ON CONTAINERS

As previously mentioned, security concerns regarding containers are the major drawback against their adoption. These security concerns have been categorized to lead the research community to study them on a common framework. Sultan et al. [5] investigated the threat model for containers across the literature and suggested four general use cases: (i) protecting containers from the applications inside, (ii) protecting containers from each other, (iii) protecting hosts from containers, (iv) protecting containers from hosts Tomar et al. [26] extended these use cases by including the Docker client as a potential target.

For our discussion, we will handle the container security challenges from two perspectives. First, the security of the application running in the container should be considered. If an application has vulnerabilities or bugs, running it in an isolated setting will not prevent the loss of availability we will experience upon those vulnerabilities getting exploited. We also need to consider the security of the containerization mechanism itself. Securing the isolation and restriction of the runtime environment results in reliable systems. Failure to do so can result in loss of confidentiality when containers operate in a multi-tenant environment through data leakages. Another class of container vulnerability emerges when the isolation mechanism of the container is broken. These vulnerabilities have been appropriately named as "container escapes". Container escapes often abuse the interface offered to container development and runtime to access the host system. In turn, those interfaces can be made accessible through the vulnerabilities in the applications themselves, allowing for arbitrary command execution inside the container environment.

Misconfiguration of containers or container runtime, as well as the default privileges container runtimes, have led to privilege escalation in the host system and the eventual compromise of it. The namespace feature of the kernel can also be exploited through namespace injec-

[1] https://sysdig.com/

tion [27], which allows a malicious container to piggyback the hosts' isolation process and see the victim container's PID space just as the host can. In this section, we will delve into one case of container escape in detail and analyze some attacks targeting the containerization process. We will base our discussion around Common Vulnerabilities and Exposures (CVE), a public effort for collecting and publishing software vulnerabilities.

CVE-2018-15664 is a vulnerability which leads to a container escape where the attacker gains free read-write access in the host system with root privileges. The vulnerable API regarding this flaw in the Docker engine is the `docker cp` call, which leverages the FollowSymlinkInScope function which allows developers to resolve paths in containers. However, Docker versions from 17.06.0-ce through 18.06.1-ce-rc2 suffer from a time-of-check to time-of-use vulnerability in FollowSymlinkInScope. Since the resolution step of the path and actually using the path are not performed sequentially, there exists a time frame where the attackers can symlink a resolved path to an arbitrary place, which includes root owned directories in the host machine [28].

CVE-2019-5736 is a vulnerability that stems from the runc binary up to version 1.0.0:rc6. runc is a container runtime that Docker as well as CRI-O, containerd and Kubernetes uses. The flaw in effected runc binary versions allow an attacker to use a malicious container to overwrite the runc binary in the host system and gain root access and privileges [29]. The only prerequisite the vulnerability requires is any command to be run as the root from the container where the said container creates a new container using an attacker-controlled image or running `docker exec` to get a shell from an already running container which gave the attacker write access previously. Prior to proper patching, this vulnerability could be prevented by using namespaces correctly and mapping the root of the host system and the container's user into different namespaces.

4. MACHINE LEARNING APPROACHES FOR CONTAINER SECURITY

Container security is handled through rule-based matching utilities, where known vulnerabilities and common misconfiguration errors are collated through human effort [30]. These utilities are adequate for catching known attacks and configuration mistakes developers make. However, they cannot detect attacks or vulnerabilities missing from their rule set. To tackle this issue, machine learning based container security solutions have been developed. In this section, we will survey container security approaches that leverage machine learning.

4.1 Intrusion detection

Zhang et al. [31] proposed an intrusion detection system for Digital Data Marketplace (DDM). The presented system utilizes the One-Class Support Vector Machines (OC-SVM) algorithm. OC-SVM is an unsupervised learning method that finds a decision boundary with maximum distance from data points, making it suitable for anomaly detection where training data is unbalanced. Similar to SVM's hyperplane, it uses a spherical boundary to separate data. They capture system calls using fixed size windows, apply preprocessing and then feed into the ML model. Besides intrusion detection, they match the output of the detection module with an attack database to decide whether the anomaly is linked to other anomalies. Their dataset contains system call data from database applications and machine learning applications which are running in containers. For database containers, they have used Sysdig. They generated traffic with Apache JMeter[2]. In addition, for unusual traffic, Metasploit for Nmap is used. For machine learning on containers, they have also used Sysdig to detect adversarial attacks during training. The trained model successfully detected 100% of the arbitrary code executions and brute force attacks with a low false positive rate. They state ROC curve values reach up to 0.995. This work is limited to system calls and does not consider any other parameters.

El Khairi et al. [15] proposed a HIDS that relies on monitoring system calls. The authors used Sysdig to collect the system calls. The novelty of their work comes from their usage of context information alongside system calls to build a graph structure to train and test their IDS. Context information includes system call arguments and recently observed system calls. The authors report that they were motivated to use context information due to the shortcomings of existing HIDS approaches. They used the LID-DS dataset [38] and extended the dataset using their contributed dataset: CB-DS, which consists of container escapes.

Here, we will explain their feature selection in detail. First, they build a graph representation of the system calls with argument information. This graph representation natively includes the recently seen system calls as well. A graph constructed for a timeframe t under benign conditions can be then used in the set of normal behavior expected during a container's normal operation. When the training is over and testing begins, any unseen vector is classified against the previously constructed benign dataset of graphs. The authors evaluated their framework on different classes of vulnerabilities and compared their approach against CDL and STIDE-BoSC.

Sever et al. [22] tackled a research gap in the container IDS literature. The authors realized that previous work focused solely on HIDS approaches that monitor system calls to train and evaluate anomaly-based IDS. This leaves anomaly-based NIDS which can leverage network traffic features such as network flow out of the picture. In order to answer whether this omission is justified or not, the authors set up an experiment environment with JMeter as the benign traffic source, a web application running in a container as the victim and the Metasploit tool as the

[2] `https://jmeter.apache.org/`

Table 1 – Overview of surveyed container intrusion detection approaches

Work	ML Method	Feature Collection	Dataset	Victim Machine	Attack Type	Monitoring
[31]	OC-SVM	n-gram	custom syscall attack dataset	CoughDB, MongoDB, static ML app	Container Privilege Escalation, Brute Force, Execution of Arbitrary Code, Adversarial ML attacks	JMeter, nmap, Sysdig
[15]	auto-encoder	system call sequence graph	LID-DS, CB-DS	Flask-python web app	Sprocket Information Leak, MySQL Auth Bypass, Release Agent Abuse, Dirty Pipe	Sysdig
[22]	REPTree, Random Tree, Random Forest, SMO	BoSC, network flow	custom syscall and network flow dataset	rConfig	OS Command Injection, SQL Command Injection	Sysdig, tcpdump
[32]	STIDE, BoSC, HMM classifiers	STIDE, BoSC	custom syscall dataset	MariaDB	Overflow, Bypass, Privilege Escalation, DoS	Sysdig
[33]	Decision Tree, Random Forest	BoSC	custom syscall dataset	MySQL	Authentication Bypass, DoS, Privilege Escalation, Integer Overflow	Sysdig
[24]	Random Forest, Isolation Forest	anonymous walk embedding	custom syscall dataset and CUI-2020	Hadoop cluster, NGINX, Apache Solr	Cryptomining, backdoor	perf
[34]	semi-supervised learning	process graph, node2vec	auditd	*contribution*	DoS, privilege escalation	auditd
[35]	variational autoencoder	time series performance event data	Container Performance Event Dataset (CEPD)	Container based big data platform	Spectre, Meltdown	ptrace, perf
[36]	auto-encoder, GAN	network traffic, system and network level performance data	VM Migration dataset created using CloudSim	2 host with 3 VM in total	Net Scan, DoS	Not mentioned
[37]	Random Forest	n-gram	ADFA-LD	Django, Httpd, MySQL, Tomcat	XSS Attack, SQL Injection, Security Policy Bypass, Remote Command Injection, Identity Bypass, Arbitrary File Read/Write	Sysdig

malicious traffic source. The authors used the Sysdig tool to gather system call traces and tcpdump to capture network traffic between the attacker machine and the victim container. They used system call data with BoSC as the feature and network flow data derived from network `.pcap` captures. In order to evaluate intrusion detection performance, the authors selected four machine learning algorithms found commonly in the container IDS literature: REPTree, random tree, random forest and SMO. After evaluating both BoSC and network flow based monitoring with those four algorithms, the authors found that network flow data yielded better performance than BoSC. However, the authors used only one victim application with only three different attacks for their dataset, putting a detriment on the generalizability of their study.

Flora et al. [32] evaluated intrusion detection performance by monitoring system calls on a containerized application by using attack injection. Their approach to this comparison is twofold: they evaluated the instruction detection performance between Docker containers, LXC and the application running on bare metal using three classifiers: BoSC, STIDE, and HMM. First, the authors decided on an application: MariaDB, running in a container for the Docker and LXC settings and standalone for the bare metal case. As is the case with attack injection approaches, they decided on the TPC-C workload for the benign traffic source. For malicious traffic, they picked 5 CVEs and used their implementations from `exploit-db.com`. The authors decided to capture every system call emitted by the containers using the sysdig tool during the experiments while they captured only MariaDB and its children's system calls for the bare metal case. Running the TPC-C workload for 24 hours with 30 minutes of malicious traffic during the benign traffic period yielded the

data required for the analysis. The authors then used the classifiers to discern between malicious and benign traffic.

During their analysis, the authors found that intrusion detection by using the methods mentioned above yielded the best overall results for the application running in the Docker container. While detection on Docker gave the highest recall across all three algorithms, BoSC performed marginally better than STIDE and wholly better than HMM. The authors also concluded that using lower epochs resulted in better detection performance and interpreted it as the models learning how to discern between the malicious and the benign traffic without learning unnecessary details. On the other hand, the authors' analysis is constrained to only one database application: MariaDB.

Cavalcanti et al. [33] compared the performance of intrusion detection systems for containers. They framed their observations under two categories: the effect of the classifier architecture and the performance of different machine learning algorithms. The authors set up an attack injection scenario where they subjected a MySQL Docker image to TPC-C benchmark for benign traffic and four different attacks from `exploit-db.com` with CVEs for malicious traffic. Overall, the authors used three classifier architectures: label encoding and one-hot encoding, sliding window with label encoding and one-hot encoding, and sliding window with BoSC.

The authors used AdaBoost, decision tree, Gaussian naive Bayes, k-nearest neighbors, multilayer perceptron, multinominal naive Bayes, random forest and support vector machine as classifiers. Gaussian naive Bayes performed the best in terms of recall, while k-nearest neighbors had the best precision out of all machine learning algorithms for the first classifier architecture. In terms of F-Measure, support vector machine had the highest performance with 83.2%. The second classifier architecture achieved the highest F-Measure of 99.4% with the random forest algorithm when the window size was 30. Both decision tree and random forest had the highest F-Measure with 99.8% for the final classifier architecture when the sliding window size was set to 30 again, albeit not much higher than other algorithms. The important takeaway from the results obtained by the authors is that the context of which the system calls appear contributes more to the detection performance than the specific machine learning algorithm chosen.

Iacovazzi and Raza [24] present a machine learning-based solution for intrusion detection in cloud containers. The proposed solution combines supervised and unsupervised learning methods, and it is designed to work at the host operating system level, using data observable at the kernel level. The solution uses a mix of random forests and isolation forests to classify container workload behaviors and detect adverse behavior within the containers. Note that random forests are supervised learning methods while isolation forests are unsupervised. First, a graph representation of the sequence of system calls are collected at the host machine's kernel level. This graph is then processed using random and anonymous walk algorithms to extract the features. This representation is fed into a random forest classifier, which is trained on normal classes and outputs a set of probabilities for whether the input belongs to each class. The probabilities are passed to a third stage, where they are used for generating anomaly scores using an ensemble of isolation forest modules, one for each normal class. Isolation forest modules are trained on datasets containing samples from the respective normal class and contaminated with samples from other normal classes. The final decision about the class of the input sample is based on the outcomes of the anomaly scores. If all anomaly scores are below a threshold, the input is classified as the class with the highest score or as an anomaly if all scores are under the threshold. If more than one score is higher than the threshold, the input is classified as an anomaly. In order to effectively capture dependencies between adjacent system calls in a sequence, which are not considered in the bag-of-system-calls approach, they use a graph-based representation. This graph representation and feature extraction process enables the effective classification of container workload behaviors and the detection of malicious behavior within the containers. Although the EoF method outperforms the SVM and LOF alternatives, there were some limitations to this approach, as it was not able to detect all attacks with a true positive rate above 0.7, namely Backdoor and SQL Injection. Moreover, the work has been conducted on Docker containers and possible attacks during container migration have not been discussed.

In their work, Pope et al. [34] introduce a new dataset derived from the Linux Auditing System, which contains both malicious and benign examples of container activity. This dataset is the first of its kind to focus on kernel-based container escapes and includes attacks such as denial-of-service and privilege escalation. The data was generated using the autoCES framework and includes partial labels identifying benign and malicious system calls over specific time intervals. However, the dataset has some limitations, including incomplete annotations and a limited number of container escape scenarios. Additionally, the selection of benign background activity in the dataset may not be comprehensive. The goal of this dataset is to be used in a semi-supervised machine learning context. For the machine learning process, they began by converting the auditd data into a process graph, which illustrated the relationships between processes. This graph was then transformed into vectors using a node embedding technique. The resulting vectors were used to train a logistic regression classifier, which was able to accurately predict whether a process was benign or malicious with an F1 score of 97%. The authors also mention that the dataset could potentially be utilized for other applications, such as training an autoencoder for anomaly detection. These results demonstrate the effectiveness of the dataset in a semi-supervised learning context.

Another work by Wang et al. [35] proposes a real-time intrusion detection system. They focus on detecting Meltdown and Spectre attacks in container environments. Spectre and Meltdown are vulnerabilities that can be exploited using cache-based side-channel attacks to access sensitive data. These vulnerabilities allow attackers to access data that is temporarily stored in the cache, which can then be extracted using cache-based side-channel attacks. In this work, to satisfy conditions for Spectre and Meltdown attacks, the scenario is designed with co-resident containers (i.e. sharing the same hardware). They designed the ContainerGuard service to watch the workflows. By monitoring, they capture hardware and software performance time-series data. After data collection, they distribute data to corresponding variational autoencoders considering the performance data category which are hardware CPU events, hardware cache events and software events. For the purpose of evaluating a method for detecting the Meltdown and Spectre attacks, a dataset called the container performance event dataset which includes 400,000 benign and 60,000 malicious data was created. The method's highest AUC score ranges from 0.90 to 0.99. In addition to the detection performance, there is no significant runtime performance overhead which is measured as approximately 4.5%.

Chakravarthi et al. [36] focused on assessing the effectiveness of anomaly detection during service and virtual migrations in cloud environments. The authors trained autoencoders and SVM on the generated dataset. They state that autoencoders perform well during VM migrations with a false positive rate below 15%. They used the reconstruction error of the autoencoder model as the anomaly score. One limitation of their work is that there is no benchmarked dataset available to test the resilience of the cloud infrastructure. They generated data samples from a simulated network and balanced them using the generative adversarial networks. These samples were classified as either anomalous or normal using the autoencoder model. However, their trained model is only able to detect anomalous traffic in a cloud environment that is similar to the one simulated in their experiments.

Clustering algorithms aim to divide the provided unlabeled data into clusters that achieve high inner similarity and outer dissimilarity. They do not rely on signatures, a description of attack classes, or labeled data, therefore for the purpose of detecting anomalies in unlabeled data, unsupervised IDS and clustering approaches are used.

To increase the effectiveness of anomaly detection in the edge computing environment, Shen et al. [37] proposed an anomaly detection framework combining cluster algorithms. The proposed framework initially identifies and classifies containers before building anomaly detection for each group. Also, they use system calls to inspect containers' behavior and perform classification and intrusion detection. They looked into eight real-world vulnerabilities, and the experiment results show that the framework increased the True Positive Rate (TPR) from 90.3% to 96.2%, and False Positive Rate (FPR) reduced from 0.61% to 0.09% compared to the traditional method. The framework utilizes Sysdig to collect system call data generated by containers, the DBSCAN cluster algorithm to classify containers in an unsupervised way, and the random forest classifier for each application category to detect anomalies. Also, they used their approach with two different detection methods. First, they used one detector for all containers. This method collects system calls from all applications without distinguishing between applications. The other method uses one detector for each container. Even though the second approach achieves better results, it incurs a significant performance cost.

Table 2 presents a succinct overview of several surveyed papers on intrusion detection in container environments, highlighting both their advantages and potential limitations. Certain papers demonstrate low false positive rates for various attacks, efficiently handle container scalability, and showcase improved performance compared to traditional baselines. They provide realistic insights into algorithm performance, offer annotated datasets and real-world simulations, and enable seamless integration into container-based big data platforms. In contrast, specific papers encounter challenges in distinguishing normal and anomalous traces in adversarial machine learning attacks or exhibit limited detection of specific vulnerabilities. Moreover, they may demonstrate limited ability to generalize, lack in-depth analysis for certain tools, or exhibit lower true positive rates for specific attack types. Some papers possess restricted scenarios within their datasets or focus exclusively on particular attacks within container-based big data platforms. Additionally, certain papers necessitate prior knowledge of application categories, posing challenges for novel and unidentified applications. Despite these limitations, the surveyed papers collectively contribute valuable insights and advancements to the field of intrusion detection in container environments.

4.2 Malware detection

Wang et al. [39] designed and implemented a malware detection framework for containerized applications. The novelty of their work comes from their approach extracting executables from containers with respect to the container's storage driver type. The authors decided to support overlay2 and aufs, the current and past recommended storage drivers respectively. With the executable in hand, the suggested framework first uses disassembled code and binary itself for fast path coarse detection using a multichannel CNN. The slow path detection is done using an LSTM-CNN with API-call sequences as the features. The authors have evaluated their implementation on 3000 malware samples acquired from VirusShare and 300 container-specific attacks against 2000 benign binaries. Even though the authors compared their framework against previous work under metrics such as precision and recall, the previous work they opted to compare to are not from the container security domain but deal with general software security. Hence, the 300 container-specific

Table 2 – Overview of surveyed intrusion detection methods: Key advantages and potential limitations

Work	Advantage	Limitation/Disadvantage
[31]	Low false positive rate for container escalation, brute force, and adversarial ML attacks.	Distinguishing between normal and anomalous traces in adversarial ML attacks can be more challenging due to weak distinctions.
[15]	Efficiently handles container scalability.	Limited detection of low syscall vulnerabilities and potential evasion by knowledgeable attackers.
[22]	Network flow-based detection outperforms BoSC representation from system calls.	Limited ability to generalize due to the use of a single vulnerable application and a small number of CVEs distributed over two CWEs.
[32]	Results provide realistic insights into algorithm performance by using representative workloads and attacks.	Approach lacks detailed FPR analysis for certain tools like Clair static analysis.
[33]	High F-Measure values indicate effective intrusion detection in multi-tenant container environments.	Lack of comparative analysis, limited sliding window size range, and limited evaluation of encoders are notable limitations.
[24]	Improved performance compared to traditional baselines such as one-class SVM and LOF models.	Lower true positive rates for certain attacks like Backdoor, SQL Injection, and Brute Force Login.
[34]	Annotated container-escape dataset and real-world edge device with simulated VM.	Limited container escape scenarios in the dataset (DoS and Privilege Escalation).
[35]	Easy integration into container-based big data platforms without hardware or kernel modifications.	Limited to detection of meltdown and spectre attacks in container-based big data platforms.
[36]	Considers the impact of VM migration on anomaly detection performance.	Focuses on detecting Net Scan (NS) and Denial of Service (DoS) attacks.
[37]	Automatic classification of containers without manual labeling using the DBSCAN algorithm.	Requires prior knowledge of application categories for building separate detection models, posing challenges for new and unknown applications.

attacks are mixed in with the rest of the 3000 malware samples and there is no particular insight presented for regular software shipped in containers and vulnerable containers.

Cryptomining malware has become a significant threat in Kubernetes, with hidden executables that uses server resources for mining. To detect and classify pods that hold cryptomining processes, Karn et al. [23] proposed that machine learning can be used together with system calls. They used several types of cryptominer images, namely Bitcoin, Bytecoin, Vertcoin, Dashcoin, and Litecoin. Also, they included healthy pods, that are MySQL, Cassandra, Hadoop, Graph, Analytics and Deeplearning. They captured system calls with a period of 1 minute for each pod. Then they leveraged n-grams to extract features. After numerous experiments they decided to set n as 35 due to its high recall rate. Following the feature extraction, four ML models which are decision tree, ensemble learning, feed-forward vanilla artificial neural network(ANN) and feedback recurrent neural network were selected to train with the data collected. The accuracy of the ensemble learning model from the Python-XgBoost library was similar on training and validation sets, 89.3% and 89.4% respectively. For feed-forward Vanilla ANN, they used the combination of Keras and Tensorflow, with the autokeras tool to tune hyperparameters. The accuracy was 81.1% on the training set, and 79.7 % on validation. Due to the nature of system calls, they are suitable for use as time-series data. Therefore, they implemented LSTM RNN. The accuracy was 79.99% on the training set and 78.90% on the validation set. Decision tree implementation with default parameter values using Python's SKLearn library achieved 99.6% accuracy on the training and 97.1% accuracy on the validation set, outperforming all other models. In addition, for better model explainability and visual representation, they used the SHAP and LIME tools.

4.3 Attack detection

Lin et al. [40] proposed an attack detection framework which consists of different layers in a pipeline in an attempt to increase detection rate while addressing false positive and lack of labeled training data issues. Their proposal has three different modules: first, they employ an unsupervised anomaly detection layer which uses an autoencoder neural network. The authors claim that the encoder and the subsequent decoder will generate results with a high reconstruction error for anomalous samples. The second layer in the pipeline uses the random forest algorithm to cross-validate edge cases and potentially

eliminate false positives. On the final layer, the authors employ an isolation forest to detect outliers and generate training labels automatically. This pipeline is fed with system call frequency vectors, acquired using Sysdig with a sampling rate of 100 milliseconds.

In order to evaluate their proposed framework, the authors applied 7 minutes worth of benign traffic onto the containers using JMeter, where applicable. At the start of the 5th minute, the authors started the attack; some attacks caused the container to crash which ended the experiment, but for the rest, the attack completed and the experiment ran until the 7th minute. The authors compared their proposed framework against CDL [41], selfpatch, a supervised random forest approach and a supervised CNN. They used 41 real world attacks with assigned CVEs, encompassing 28 applications. They used containerized applications with application vulnerabilities, not container-specific attacks.

Lin et al. [41] presented a classified distributed learning framework, namely CDL, to detect anomalies in containerized applications. The framework achieves anomaly detection in four major steps: system call feature extraction, application classification, system call data grouping, classified learning, and detection. They process raw system call traces into a stream of frequency vectors, and these extracted feature vectors are used to identify applications. For the identification of applications, they utilize the random forest classifier [42]. When this process identifies the containers of the same application, the framework makes a system call data grouping to append the frequency vector traces of different containers and uses them for model training and attack detection. Lastly, for anomaly detection, the unsupervised model uses autoencoders. The authors investigated 33 real-world vulnerabilities documented in the Common Vulnerabilities and Exposures (CVE) database, and the results show that CDL can detect 31 out of 33 attacks. Also, they inspected the system run time, and the data indicates that CDL is lightweight and suitable for detecting attacks in real time under real-world circumstances.

4.4 Anomaly detection

Gantikow et al. [43] investigated the behavior of containers using neural networks to detect anomalies. The authors present two approaches for anomaly detection based on system call traces. First, system call distributions are used to detect anomalies. A one-layer LSTM network is trained to predict the system call distribution at time $t+1$ based on distribution at time t. The second approach is a neural network using file/directory paths for anomaly detection. Their method is based on training a neural network to predict the following file system path based on the most recent file system path used by a system call. The proposed neural network consists of a word embedding layer, followed by LSTM layers which are designed to learn to predict the following file system path based on the vector representation of the current one. After a prediction is made by this neural network, the actual file path and predicted path are compared to detect anomalies.

Wang et al. [44] proposed an unsupervised anomaly detection framework. The authors initially acquired system call sequences using the ptrace tool. Then, they used the word2vec technique to map each system call within their context from the sequences into a fixed size vector. These vectors are used sequentially for the rest of the author's instruction detection framework. At the final layer of their framework, the system detects anomalies through reconstruction error. For evaluation, the authors employed the UNM system call sequences dataset. They also extended it with system call sequences gathered during a sqlmap attack on a container running MySQL, as well as three different container escape attacks. The dataset they used for evaluation consisted of 0.63% anomalous traces with benign samples as the rest. Overall, their approach yielded 90% accuracy and an F1 score of 90.75%.

Castanhel et al. [45] present an approach for using system calls to detect anomalies in containerized systems. The authors focus on how the size of the window impacts the results through the implementation of a sliding window technique. In their implementation, the authors collected a dataset of system calls by running strace on the host machine, outside the container, from a variety of containerized applications and used machine learning techniques to train a model to classify normal and anomalous system calls based on this dataset. The dataset used in the study consisted of 50 traces of system calls, with half representing normal behavior and the other half representing anomalous behavior. The normal behavior traces consisted of five different types of expected interactions with the WordPress application, while the anomalous behavior traces consisted of five different types of attacks focusing on cross-Site Scripting (XSS) and Remote Code Execution (RCE).

The experiments in the study were conducted on a Linux host using Docker. The collected system calls were divided into four groups, with the first group containing the most dangerous system calls that alter system behavior. The last group contained harmless system calls that primarily query to get system behavior rather than issuing commands. A sliding window technique was used to analyze data from various sources and four algorithms were applied using seven different window sizes. They tested both with all data and the data without harmless system calls and found that the model was able to accurately detect anomalies in the system calls of containerized applications, with an average accuracy of over 90%. The study concluded that system calls can be an effective means of detecting anomalies in containerized systems but also mentioned the fact that their work does not contain all calls available in current systems. Also, by completing tasks using a variety of containers with different applications instead of just the WordPress application with additional plugins, the dataset would have been more diverse, allowing for a more comprehensive evaluation of the system's efficacy.

Table 3 – Overview of surveyed anomaly detection in container approaches

Work	ML Model	Data Used	Collecting Method
[43]	LSTM	system call, file/directory path	Sysdig
[44]	BiLSTM	system call	ptrace
[45]	KNN, RF, MLP, AB	system call	strace
[46]	LSTM auto-encoder	system call	Sysdig
[47]	KNN, k-means SOM	system call	CoreOS clair, Sysdig, JMeter
[48]	KNN, SVM, NB, RF	performance monitoring data	cAdvisor, Heapster
[25]	DCRNN	RPC traffic	RPC chain clustering
[49]	Restricted Boltzman Mahcine	user and system defined security profile, automated NIST violations, runtime security profile	python script
[50]	*contributed*	system call, network, I/O activities	JMeter, Sysdig
[51]	LR, NB, SVM, RF, XGB	security related config documents	BeautifulSoup, NLTK
[52]	SARIMA, HMM, LSTM, auto-encoder	system metrics (streaming data)	Prometheus

Cui and Umphress [46] created an open-source dataset for the observation of system calls. They chose to use a classic LSTM model as the baseline classifier for the unsupervised classification task. The reason behind choosing LSTM is that it has the ability to remember and use knowledge from previous batches, which makes it suitable for anomaly detection. In the experiments, a total of 42 models were trained using different combinations of configurations, including seven different window sizes, three different feature sets, and two normalization methods. These models were then tested on seven different attacks, with six different confidence levels applied, resulting in a total of 1764 entries. Overall the model achieved over 90% accuracy for brute force login, meterpreter, malicious script and remote shell attacks. However, its accuracy on Docker escape attacks was only 76.27%. Moreover, it was observed that the proposed framework was only evaluated using an offline dataset and a single application. Besides, while the work successfully demonstrated the potential for unsupervised introspection, it is necessary to expand the dataset to include multiple applications to see its potential in different contexts.

Tunde et al. [47] presented a combination of static and dynamic anomaly detection schemes to detect security vulnerabilities for containers. They conducted a study on static and dynamic vulnerability detection strategies using 28 common real-world security vulnerabilities discovered in Docker Hub images. Firstly, they used CoreOS Clair, an open-source static analysis engine that scans containers layer-by-layer for known vulnerabilities using CVE databases. Afterwards, they investigated dynamic detection schemes using different unsupervised machine learning algorithms. These machine learning algorithms are selected to address the following unique challenges of container security:

1. Containers are short-lived, so the detection algorithms cannot use large amounts of training data.

2. Containers are highly dynamic; thus, the detection algorithms cannot make any assumptions about the application or attack behavior in advance.

3. The detection algorithms should be able to detect vulnerabilities with a low overhead.

These challenges led the authors to use lightweight unsupervised anomaly detection schemes such as k-nearest neighbor, k-means clustering, KNN combined with Principal Component Analysis (PCA), and Self-Organizing Map (SOM). Their comparison between different exploit detection schemes was based on the following metrics: detection coverage, false positive rate, and lead time. The metrics indicate if each approach can detect vulnerabilities, how accurately they can achieve detection and how quickly they can detect attacks, respectively. The KNN algorithm was used to perform outlier detection. Because the presence of noise in the feature data prevents the KNN algorithm from achieving high accuracy, they used KNN with PCA. While the KNN algorithm had a detection coverage rate of 32.14%, PCA + KNN had a slightly better detection coverage rate of 35.71%. The k-means approach achieved a 67.86% detection coverage rate. The Self-Organizing Map (SOM) approach over system call time vectors (SOM time) detected 75% of the vulnerabilities, while the SOM approach over system call frequency vectors (SOM frequency) detected 79% of the vulnerabilities. Therefore, the SOM approach accomplished the highest detection coverage. At the false positive rate comparison, again the SOM approach achieved the lowest false positive rate, with 1.7% for the SOM frequency and 1.9% for the SOM time. It is followed by the k-means clustering approach with 7.67%. Moreover, KNN and KNN with

PCA obtain the highest false positive rates with 9.92% and 9.88%, respectively. Finally, the SOM approach attained the largest detection lead time, with 28.7 seconds for SOM frequency and 25.8 seconds for SOM time. On the other hand, KNN, KNN with PCA, and k-means achieved 0.57, 1, and 0.36 seconds respectively. The authors also stated that combining static and dynamic schemes can increase the detection coverage rate to 86%. They concluded that static analysis for container security is insufficient and dynamic anomaly detection schemes can achieve a high detection coverage rate with a low false positive rate.

Du et al. [48] used different supervised machine-learning algorithms to detect anomalies in container-based microservices. The proposed anomaly detection system consists of three modules: the monitoring module, the data processing module, and the fault injection module. First, the monitoring module is used to collect real-time performance monitoring data from the target system. In the paper, the authors focused on container and microservice monitoring, and the term "container" was used to refer to a collection of containers constituting one complete microservice. Secondly, the data processing module is used to analyze this data and detect anomalies. They determine whether a container performs well by gathering and processing its performance data, just as they determine whether a microservice is abnormal by gathering and processing the performance data of all related containers. After classifying if a microservice has an anomaly, the anomaly detection system finds the anomalous container. In order to detect anomalies, they use supervised machine learning algorithms such as SVM, random forest, naive Bayes and KNN. Also, to find the container that caused an anomaly, time series analysis is used. Lastly, the fault injection module simulates service faults (CPU consumption, memory leak, network package loss, and network latency increase) and collects datasets of performance monitoring data. These datasets are used to train machine learning models to validate the anomaly detection performance. The validation results show that random forest and KNN classifiers achieved satisfying results using each dataset. Furthermore, SVM performs the worst since it does not work well on datasets with multiple classes.

Remote Procedure Calls (RPC) allow components in a distributed cluster of applications to invoke each other's functions (procedures) seamlessly, as if those functions are owned by the invoking application. The network layer between the components is abstracted as a result. Chen et al. [25] suggested using RPCs as an alternative to monitoring system calls, since RPCs are required for meaningful interaction between a distributed cluster's components just like system calls being required for worthwhile operations within an application. They handled RPCs as RPC chains, a sequence of RPCs that depend on each other and appear in order during common operations. The authors found that representing RPC chains as directed weighted graphs suits their use case well. They represented nodes as RPCs, edges and weights as the dependency between different RPCs, and labeled nodes with the number of times that particular RPC was invoked.

To learn regular RPC traffic and predict anomalous RPC traffic, the authors first trained the DBSCAN [53] clustering algorithm to acquire the RPC chains. The authors then trained a DCRNN model to predict the traffic model from previously observed RPC traffic. By using mean absolute error and variants, they managed to label anomalous traffic when observed RPC chains deviated from the expected traffic in their case study which was performed on a Kubernetes cluster with "billions of daily active users" [25] and RPC traffic that spans two weeks.

Kamthania [49] presents a deep learning algorithm for detecting malicious patterns in individual container instances. The algorithm was designed to be easily applied to any container platform that adheres to the Open Container Initiative (OCI) standard. The algorithm utilizes a Gaussian-Bernoulli restricted Boltzmann machine. A Restricted Boltzmann Machine (RBM) is a type of neural network that is used for unsupervised learning. RBMs are composed of a visible layer, which encodes the input data, and a hidden layer, which learns features from the input data. RBMs are trained using an energy-based model, where the energy of a configuration of the visible and hidden units is minimized. RBMs are often used for tasks such as dimensionality reduction and collaborative filtering. Gaussian-Bernoulli RBMs are a variant of RBMs that can handle continuous-valued data, rather than just binary data, in the visible layer.

By using RBM, they create a container profile based on the configuration of the containers and extract behavioral statistics at runtime. The algorithm then uses automated NIST container security rules to identify any security violations for the container under test and applies a machine learning algorithm to build a complete security profile for the container. In their results they included the following attack types: unbounded network access from containers, insecure runtime configurations, rogue containers, improper user access rights, embedded clear texts.

KubAnomaly, a system that offers security monitoring capabilities for anomaly detection on the Kubernetes orchestration platform was proposed by Tien et al. [50]. The aim of this system is to improve Docker security, which is compatible with Kubernetes. KubAnomaly provides a security monitoring module with customized rules in Sysdig and observes the internal activities of containers, system calls, I/O activities, and network connections. Since monitoring too many events would result in a large overhead, they selected four system call categories which are file I/O, network I/O, scheduler and memory. A neural network model was created to classify multiple types of anomalous behavior, such as injection attacks and Denial-of-Service (DoS) attacks. Three different datasets were used to train the ML models. These three different datasets include private data, a public data called CERT and real-world experimental data to evaluate the system accuracy and performance. Further explanation of the datasets is available in Section 4.6

The proposed anomaly classification model is organized into four steps. They begin by monitoring log data from their agent service, which collects monitor logs from Docker-based containers. After obtaining the raw monitor logs, they extract features to train their models. The next step is data normalization for fast convergence and improved accuracy. Lastly, they construct the anomaly classification model using four fully-connected layers, and for the backend, they use Keras and Tensorflow. The results show that KubAnomaly is able to detect many abnormal behaviors.

Mubin et al. [51] focused on configurations of container orchestrators. A container orchestrator itself should be correctly configured to provide security for all other managed containers. They introduced a new method that uses keywords and learning to capture knowledge about configurations which was not studied before. The module created, namely KGSecConfig, aims to create a Knowledge Graph for Configuration (KGConfig) of various platforms, cloud providers, and tools used in container orchestration to organize scattered data. They extracted information from documentation files and created entities. Between these entities, there exist several relationships such as "hasDefault", "hasArgument", "hasType", "hasOption", and "hasDescription". This representation is used to identify the configuration syntax and formulate keyword-based rules for estimating the relevancy of security documents with configuration.

In order to train a supervised learning model to extract configuration concepts from documents, a labeled dataset was needed. Since no labeled dataset existed, 3,300 sentences were labeled by two authors according to the four configuration concepts. 3,032 sentences that were agreed upon by both those labelling were used for training the model to reduce labeling bias. Five machine learning classifiers, logistic regression, naive Bayesian, support vector machines, random forest, and extreme gradient boosting were selected for the learning-based models, and various features such as TF-IDF-based word level, character level, and combination of word and character level, NLP features, were considered. The optimal traditional ML models were selected using Bayesian optimization and average Matthews correction coefficient with early stopping criteria. Breadth-First-Search algorithm was used to identify the configuration argument and update the KGConfig. Accuracies of the LR, NB, SVM, RF, XGB were 0.94, 0.82, 0.88, 0.76, 0.93 respectively. The model's results showed that KGSecConfig is effective in automating the mitigation of misconfigurations.

Kosinska and Tobiasz [52] proposed a system, namely, Kubernetes Anomaly Detector (KAD), for detecting anomalies in a Kubernetes cluster. KAD uses various machine learning models to achieve high accuracy. Their solution differs from other solutions in using different machine learning models that facilitate detecting different types of anomalies. The KAD system chooses the appropriate model for detection; thus, different models can be matched to different data types. These models are SARIMA, HMM, LSTM and autoencoder. SARIMA and HMM are derived from traditional time series and statistical models. In their experiments, they trained the models on the Numenta Anomaly Benchmark (NAB) dataset. They selected two types of data streams: the first stream is artificially generated, and the second contains data presenting CPU utilization collected from AWS Cloudwatch.

The results show that statistical models (SARIMA and HMM) achieve higher results on the artificial data and the LSTM and autoencoder perform better on AWS Cloudwatch data. Furthermore, the experiments demonstrate that the real-time anomaly detection capabilities of the KAD system can be successfully deployed in a Kubernetes cluster. However, the KAD system allows anomaly detection to be performed on one metric at a time. Hence, for more complex cases, multivariate models may be needed.

Table 4 provides a concise summary of the surveyed papers in the field of intrusion detection, highlighting their respective advantages and potential limitations. Each work offers unique strengths in addressing container security challenges. For example, some papers leverage neural networks to analyze file system paths, providing an additional layer of detection. Others propose innovative network architectures that effectively capture long-term dependencies in sequential data, making them suitable for analyzing and detecting anomalies in time-series data. However, it is important to consider the limitations of these approaches. Some works have a narrower focus, evaluating their methods primarily in specific scenarios, which may limit their ability to generalize. Others may face challenges in detecting certain types of attacks due to similarities in system call traces. Additionally, there are limitations related to the applicability of proposed algorithms to different programming languages or system architectures, the need for ongoing effort in maintaining and updating models in evolving systems, and subjective evaluation of extracted configuration knowledge. Understanding these advantages and limitations is crucial for selecting the most appropriate intrusion detection method for specific container-based environments.

4.5 Inter-container security

Deng et al. [54] tackled the secure placement of containers in a cloud setting where multiple residents share the same host. The motivation for their work comes from the authors' findings that container placement strategies do not consider the security of co-resident containers. As mentioned in Section 3, container vulnerabilities pose risks to the container running on the same host as well as the host system. Deng et al. considered the whole placement challenge as a series of placement decisions. This allowed the authors to reframe the problem as an optimization task. The authors then used Deep Reinforcement Learning (DRL) with the encoded placement policy as the input. DRL is a branch of machine learning that uses reinforcement learning and deep learning

Table 4 – Overview of surveyed anomaly detection methods: Key advantages and potential limitations

Work	Advantage	Limitation/Disadvantage
[43]	The use of neural networks to analyze file system paths provides an additional layer of detection.	The evaluation of the approach is primarily focused on specific scenarios, such as simultaneous execution of the normal application and the anomaly (mimicry attack)
[44]	The proposed BiLSTM-VAE network can effectively capture long-term dependencies in sequential data, making it suitable for analyzing anomalies in time-series data.	The provided information does not elaborate on the diversity and representativeness of the datasets used for evaluating the proposed approach
[45]	The paper explores the feasibility of online anomaly detection, where sliding windows are used to detect attacks in real time.	The paper highlights the need to improve the recall rate of the approach to reduce the number of false negatives.
[46]	The framework utilizes an LSTM autoencoder as the baseline classifier, which has the capability to memorize and utilize knowledge from previous batches of system calls.	The classifier may face challenges in detecting certain attacks, such as SQL injection and SQL misbehavior, where the system call traces are similar except for the different keywords used.
[47]	Extracted key code patterns guide developers in identifying vulnerable code segments and improving coding practices.	The paper focuses on vulnerabilities in Java-based cloud server systems, which may limit its applicability to other programming languages or system architectures
[48]	Integration of a fault injection module facilitates simulation of system conditions, aiding ML model training and performance assessment.	The approach assumes that there is only one anomalous container at the same time within a microservice
[25]	Models incorporate GCNs and diffusion processes to capture dynamic RPC traffic patterns, enhancing prediction accuracy.	Maintaining and updating independent models for each RPC chain pattern in an evolving microservice system requires ongoing effort.
[49]	Identified malicious data patterns serve as targeted payloads for effective container platform penetration testing.	Applicability of the proposed algorithm to other container platforms requires further evaluation and adaptation beyond Docker.
[50]	KubAnomaly achieves 96% accuracy in container anomaly detection, enhancing container security significantly.	KubAnomaly agent's privilege permission and open port pose potential security risks and vulnerability similar to the underlying gRPC framework.
[51]	KGSecConfig can automatically detect inconsistencies in the configuration knowledge by comparing it with official documentation.	Subjectivity in evaluating the quality of extracted configuration knowledge introduces a potential limitation.
[52]	Ensemble of models enhances accuracy and performance by leveraging diverse strengths in different data patterns.	KAD's limitation to univariate models restricts its ability to simultaneously analyze multiple metrics in complex cases.

principles. Nguyen et al. [55] state that DRL is suited to solve complex, dynamic, and high-dimensional cyber-defense problems. Deng et al.'s model outputs the placement decision for the given container at every time step and the reward function is a formula with a trade-off between security and workload balancing. With their evaluation, the authors find their model to perform better than existing strategies in terms of workload balance while keeping security in mind.

Li et al. [56] proposed a defensive deception framework for container-based clouds. Their approach generates an adversarial model, decoy placement strategy, and decoy routing tables using a DRL algorithm. First, they developed an adversarial model, namely the System Risk Graph (SRG). SRG extracts risks and threats in the container-based cloud and includes overall risks and vulnerabilities from the application to the visualization layer. Secondly, SRG_t, which is the system risk graph of the cloud at time slot-t, is sent to input neurons of the DRL agent. The DRL algorithm generates an ideal decoy placement strategy to decide optimal topological locations and types on digital decoys' assets. Moreover, the performance of the placement strategy is used as the reward data to train and update the DRL agent. This feature enables the DRL agent to evolve with the dynamic cloud. Therefore, their method is adaptive and fully interacts with the dynamic environ-

ment. Lastly, the determined placement strategy and deceptive routing for the decoy are sent to the orchestration platform. As a result, the proposed framework increases the detection ratio on the random-walker attack by 30.69% and the persistent attack by 51.10%.

Using Genetic Algorithm (GA), Kong et al. [57] proposed a Secure Container Deployment Strategy (SecCDS) to defend against co-resident attacks in container clouds. SecCDS reduces co-residency by 50% compared with existing strategies by coordinating the placement and migration of containers to separate attackers and victims on different Physical Machines (PMs). In their paper, they define two metrics to describe the deployment and co-residency of container clouds. Deployment Matrix (DM) represents the correspondence between container and PM, and Coresidency Matrix (CM) describes co-residency between different tenants in the cloud. Later, they develop a deployment strategy by genetic algorithm that can detect relational aggression between different tenants in real time and dynamically migrate containers, effectively preventing co-residency. In this implementation, a container must only be deployed in a unique position and belong to a real tenant. Thus, the authors offered a genetic mechanism with altered crossover and mutation operations of traditional genetic algorithm by changing some mutation operation steps and proposed a new individual learning mechanism. In addition, they utilized Simulated Annealing (SA) to do a neighborhood search for each individual in GA, which helps the algorithm reach the optimal global solution.

4.6 Dataset for container security

Chakravarthi et al. [36] used the CloudSim 5.0 environment and collected traffic data. They augmented the data using a generative adversarial network. The dataset contains TCP traffic. Since the work focuses on anomaly detection in scenarios where VM migration occurs, to detect particularly volume-based attacks, by analyzing the traces of TCP streams is beneficial since these types of attacks tend to consume an excessive amount of bandwidth compared to normal traffic. The collected data contains a wide variety of features, some of which are CPU, network and memory usages. In addition, they gathered information about the status of the network flow. To solve the data imbalance problem, the authors decided to augment data based on the collected samples. For this task, they selected GAN [36] rather than restricted Boltzmann machines or variational autoencoders. Creating new samples becomes a challenging task when the data has many variables or features. During simulation, they introduced Net Scan and DoS attacks, but the proposed work does not include information about the statistical properties of the dataset, such as the percentage of data collected for each metric, which would be useful in understanding the characteristics of the data.

Cui and Umphress [46] created an open-source dataset for the observation of system calls. Scripts for data generation are available in their public repository. The work aims to improve upon previous datasets used for detecting anomalies in computer systems by addressing some of their limitations. These include focusing on network traces rather than internal system behaviors, limited scope and coverage in system call-based datasets, and a lack of clear descriptions of benign behaviors and indications of system activity. Additionally, at that time, the authors stated that none of the previous datasets were explicitly designed for containerized systems. Brute force login, simple remote shell, malicious Python script, SQL misbehavior, SQL injection, Docker escape and other selected malware types were included in the dataset. They captured 7,144,780 benign system calls which constitutes the majority of the dataset. As a limitation, the sample application used in this study was a MySQL database.

Tien et al. [50] used three different datasets to train the supervised machine learning model for the security system called KubAnomaly. These datasets are private data, public data called CERT, and real-world experimental data to evaluate the system's accuracy and performance. First, they used a simple dataset and a complex dataset. These datasets contain two parts: 80% for training and 20% for testing, and both datasets include normal and abnormal samples. Normal samples include several types of web services run in containers. They used JMeter to simulate user login behavior. The abnormal samples include two types of attacks aimed at compromising the web service. They used Owasp Zap to simulate a hacker's attempt to attack the container and JMeter to simulate a DoS attack. The complex dataset also contains both normal and abnormal sample types. The complex dataset includes other hacker tools such as sqlmap. KubAnomaly achieves over 98% accuracy with the simple dataset and 96% accuracy with the complex dataset. CERT contains various types of log data, inclusive of email and device data, but it does not include system call log data. This dataset does not have any labeling; therefore, they used feature extraction and unsupervised learning to classify abnormal user behavior. Finally, in order to attract hackers, the authors developed an online web service with vulnerabilities and used KubAnomaly to identify abnormal behaviors and record the attack events.

In their experiment for the selection of anomaly detection model, Kosinska and Tobiasz [52] used the Numenta Anomaly Benchmark (NAB) dataset to train the models. They chose two different sorts of data streams: the first is artificially generated, and the second contains data presenting CPU utilization collected from AWS Cloudwatch.

5. CONCLUSION

This survey has provided a comprehensive overview of container security in 5G environments and the potential of machine learning-based methods to address the challenges posed by increased connectivity. The integration of ML into security systems has the potential to enhance intrusion and malware detection, anomaly detec-

tion, attack detection, and inter-container security within container clusters, making it a powerful tool in the fight against cyberattacks. The use of ML-based methods for container security in 5G environments has the potential to revolutionize the way we protect and secure our digital assets. Further research and development in this field is needed to fully realize the potential of ML-based approaches for container security in 5G environments.

Future work to advance container security in 5G environments based on this survey include addressing the ability to interpret and explain, detecting and mitigating bias in AI models, and conducting real-world validation. Improving the ability to interpret and explain involves developing techniques to enhance transparency and understanding of AI-based container security methods. Detecting and mitigating bias in AI models ensures fairness and prevents discriminatory outcomes. Real-world validation through empirical studies and collaborations with industry partners evaluates the performance and identifies potential challenges, refining the practical implementation. Pursuing this future work will enhance container security in 5G environments, fostering robust and trustworthy AI-based solutions.

This survey aims to contribute to the growing body of knowledge in this field and provide a valuable resource for researchers, practitioners, and decision-makers working in container security and 5G networks.

ACKNOWLEDGEMENT

This research has been supported by TÜBA GEBİP and the TÜBİTAK 3501 Career Development Program under grant number 120E537. However, the entire responsibility of the publication belongs to the owners of the research. The financial support received from TÜBİTAK does not mean that the content of the publication is approved in a scientific sense by TÜBİTAK.

REFERENCES

[1] Sébastien Vaucher, Rafael Pires, Pascal Felber, Marcelo Pasin, Valerio Schiavoni, and Christof Fetzer. "SGX-Aware Container Orchestration for Heterogeneous Clusters". In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. July 2018, pp. 730–741. DOI: 10.1109/ICDCS.2018.00076.

[2] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. "The Pains and Gains of Microservices: A Systematic Grey Literature Review". In: *Journal of Systems and Software* 146 (Dec. 2018), pp. 215–232. ISSN: 0164-1212. DOI: 10.1016/j.jss.2018.09.082. (Visited on 01/14/2023).

[3] Kuljeet Kaur, Tanya Dhand, Neeraj Kumar, and Sherali Zeadally. "Container-as-a-Service at the Edge: Trade-off between Energy Efficiency and Service Availability at Fog Nano Data Centers". In: *IEEE Wireless Communications* 24.3 (June 2017), pp. 48–56. ISSN: 1558-0687. DOI: 10.1109/MWC.2017.1600427.

[4] Mbarka Soualhia and Fetahi Wuhib. "Automated Traces-based Anomaly Detection and Root Cause Analysis in Cloud Platforms". In: *2022 IEEE International Conference on Cloud Engineering (IC2E)*. Sept. 2022, pp. 253–260. DOI: 10.1109/IC2E55432.2022.00034.

[5] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. "Container Security: Issues, Challenges, and the Road Ahead". In: *IEEE Access* 7 (2019), pp. 52976–52996. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2911732.

[6] Ali Bou Nassif, Manar Abu Talib, Qassim Nasir, and Fatima Mohamad Dakalbab. "Machine Learning for Anomaly Detection: A Systematic Review". In: *IEEE Access* 9 (2021), pp. 78658–78700. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3083060. (Visited on 12/12/2022).

[7] Pilla Vaishno Mohan, Shriniket Dixit, Amogh Gyaneshwar, Utkarsh Chadha, Kathiravan Srinivasan, and Jung Taek Seo. "Leveraging Computational Intelligence Techniques for Defensive Deception: A Review, Recent Advances, Open Problems and Future Directions". In: *Sensors* 22.6 (Mar. 2022), p. 2194. ISSN: 1424-8220. DOI: 10.3390/s22062194. (Visited on 12/12/2022).

[8] Zhiheng Zhong, Minxian Xu, Maria Alejandra Rodriguez, Chengzhong Xu, and Rajkumar Buyya. "Machine Learning-based Orchestration of Containers: A Taxonomy and Future Directions". In: *ACM Computing Surveys* 54.10s (Sept. 2022), 217:1–217:35. ISSN: 0360-0300. DOI: 10.1145/3510415. (Visited on 12/05/2022).

[9] Ann Yi Wong, Eyasu Getahun Chekole, Martin Ochoa, and Jianying Zhou. *Threat Modeling and Security Analysis of Containers: A Survey*. Nov. 2021. DOI: 10.48550/arXiv.2111.11475. arXiv: arXiv:2111.11475. (Visited on 12/10/2022).

[10] René Peinl, Florian Holzschuher, and Florian Pfitzer. "Docker Cluster Management for the Cloud - Survey Results and Own Solution". In: *Journal of Grid Computing* 14.2 (June 2016), pp. 265–282. ISSN: 1572-9184. DOI: 10.1007/s10723-016-9366-y. (Visited on 01/14/2023).

[11] Zhuping Zou, Yulai Xie, Kai Huang, Gongming Xu, Dan Feng, and Darrell Long. "A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest". In: *IEEE Transactions on Cloud Computing* 10.1 (Jan. 2022), pp. 134–145. ISSN: 2168-7161. DOI: 10.1109/TCC.2019.2935724.

[12] Mashal Abbas, Shahpar Khan, Abdul Monum, Fareed Zaffar, Rashid Tahir, David Eyers, Hassaan Irshad, Ashish Gehani, Vinod Yegneswaran, and Thomas Pasquier. "PACED: Provenance-based Automated Container Escape Detection". In: *2022 IEEE International Conference on Cloud Engineering (IC2E)*. Sept. 2022, pp. 261–272. DOI: 10.1109/IC2E55432.2022.00035.

[13] Robert W. Shirey. *Internet Security Glossary, Version 2*. Request for Comments RFC 4949. Internet Engineering Task Force, Aug. 2007. DOI: 10.17487/RFC4949. (Visited on 06/20/2022).

[14] William Stallings and Lawrie Brown. *Computer Security: Principles and Practice*. Third. Pearson, 2014. ISBN: 978-0-13-377392-7.

[15] Asbat El Khairi, Marco Caselli, Christian Knierim, Andreas Peter, and Andrea Continella. "Contextualizing System Calls in Containers for Anomaly-Based Intrusion Detection". In: *Proceedings of the 2022 on Cloud Computing Security Workshop*. CCSW'22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 9–21. ISBN: 978-1-4503-9875-6. DOI: 10.1145/3560810.3564266. (Visited on 12/10/2022).

[16] Siddharth Srinivasan, Akshay Kumar, Manik Mahajan, Dinkar Sitaram, and Sanchika Gupta. "Probabilistic Real-Time Intrusion Detection System for Docker Containers". In: *SSCC*. 2018.

[17] Dae-Ki Kang, D. Fuller, and V. Honavar. "Learning Classifiers for Misuse and Anomaly Detection Using a Bag of System Calls Representation". In: *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*. June 2005, pp. 118–125. DOI: 10.1109/IAW.2005.1495942.

[18] Amr S. Abed, T. Charles Clancy, and David S. Levy. "Applying Bag of System Calls for Anomalous Behavior Detection of Applications in Linux Containers". In: *2015 IEEE Globecom Workshops (GC Wkshps)*. Dec. 2015, pp. 1–5. DOI: 10.1109/GLOCOMW.2015.7414047.

[19] A. S. Abed, T. Clancy, and David S. Levy. "Intrusion Detection System for Applications Using Linux Containers". In: *STM* (2015). DOI: 10.1007/978-3-319-24858-5_8.

[20] Amr S. Abed, Mohamed Azab, Charles Clancy, and Mona S. Kashkoush. "Resilient Intrusion Detection System for Cloud Containers". In: *International Journal of Communication Networks and Distributed Systems* 24.1 (2020), p. 1. ISSN: 1754-3916, 1754-3924. DOI: 10.1504/IJCNDS.2020.103857. (Visited on 12/11/2022).

[21] Martin Max Röhling, Martin Grimmer, Dennis Kreubel, Jorn Hoffmann, and Bogdan Franczyk. "Standardized Container Virtualization Approach for Collecting Host Intrusion Detection Data". In: *2019 Federated Conference on Computer Science and Information Systems (FedCSIS)*. Sept. 2019, pp. 459–463. DOI: 10.15439/2019F212.

[22] Yigit Sever, Goktug Ekinci, Adnan Harun Dogan, Bugra Alparslan, Abdurrahman Said Gurbuz, Vahab Jabrayilov, and Pelin Angin. "An Empirical Analysis of IDS Approaches in Container Security". In: *2022 International Workshop on Secure and Reliable Microservices and Containers (SRMC)*. Sept. 2022, pp. 18–26. DOI: 10.1109/SRMC57347.2022.00007.

[23] Rupesh Raj Karn, Prabhakar Kudva, Hai Huang, Sahil Suneja, and Ibrahim M. Elfadel. "Cryptomining Detection in Container Clouds Using System Calls and Explainable Machine Learning". In: *IEEE Transactions on Parallel and Distributed Systems* 32.3 (Mar. 2021), pp. 674–691. ISSN: 1558-2183. DOI: 10.1109/TPDS.2020.3029088.

[24] Alfonso Iacovazzi and Shahid Raza. "Ensemble of Random and Isolation Forests for Graph-Based Intrusion Detection in Containers". In: *2022 IEEE International Conference on Cyber Security and Resilience (CSR)*. July 2022, pp. 30–37. DOI: 10.1109/CSR54599.2022.9850307.

[25] Jiyu Chen, Heqing Huang, and Hao Chen. "Informer: Irregular Traffic Detection for Containerized Microservices RPC in the Real World". In: *High-Confidence Computing* 2.2 (June 2022), p. 100050. ISSN: 2667-2952. DOI: 10.1016/j.hcc.2022.100050. (Visited on 06/05/2022).

[26] Aparna Tomar, Diksha Jeena, Preeti Mishra, and Rahul Bisht. "Docker Security: A Threat Model, Attack Taxonomy and Real-Time Attack Scenario of DoS". In: *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. Jan. 2020, pp. 150–155. DOI: 10.1109/Confluence47617.2020.9058115.

[27] Wonjun Lee and Mohammad Nadim. "Kernel-Level Rootkits Features to Train Learning Models Against Namespace Attacks on Containers". In: *2020 7th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2020 6th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*. Aug. 2020, pp. 50–55. DOI: 10.1109/CSCloud-EdgeCom49738.2020.00018.

[28] Aleksa Sarai. *Oss-Sec: CVE-2018-15664: Docker (All Versions) Is Vulnerable to a Symlink-Race Attack*. https://seclists.org/oss-sec/2019/q2/131. May 2019. (Visited on 01/16/2023).

[29] Aleksa Sarai. *Oss-Security - CVE-2019-5736: Runc Container Breakout (All Versions)*. Feb. 2019. (Visited on 01/16/2023).

[30] Francesco Minna, Fabio Massacci, and Katja Tuma. "Towards a Security Stress-Test for Cloud Configurations". In: *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*. July 2022, pp. 191–196. DOI: 10.1109/CLOUD55607.2022.00038.

[31] Lu Zhang, Reginald Cushing, Cees de Laat, and Paola Grosso. "A Real-Time Intrusion Detection System Based on OC-SVM for Containerized Applications". In: *2021 IEEE 24th International Conference on Computational Science and Engineering (CSE)*. Shenyang, China: IEEE, Oct. 2021, pp. 138–145. ISBN: 978-1-66541-660-3. DOI: 10.1109/CSE53436.2021.00029. (Visited on 01/09/2023).

[32] José Flora, Paulo Gonçalves, and Nuno Antunes. "Using Attack Injection to Evaluate Intrusion Detection Effectiveness in Container-based Systems". In: *2020 IEEE 25th Pacific Rim International Symposium on Dependable Computing (PRDC)*. Dec. 2020, pp. 60–69. DOI: 10.1109/PRDC50213.2020.00017.

[33] Marcos Cavalcanti, Pedro Inacio, and Mario Freire. "Performance Evaluation of Container-Level Anomaly-Based Intrusion Detection Systems for Multi-Tenant Applications Using Machine Learning Algorithms". In: *The 16th International Conference on Availability, Reliability and Security*. ARES 2021. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1–9. ISBN: 978-1-4503-9051-4. DOI: 10.1145/3465481.3470066. (Visited on 05/08/2022).

[34] James Pope, Francesco Raimondo, Vijay Kumar, Ryan McConville, Rob Piechocki, George Oikonomou, Thomas Pasquier, Bo Luo, Dan Howarth, Ioannis Mavromatis, Pietro Carnelli, Adrian Sanchez-Mompo, Theodoros Spyridopoulos, and Aftab Khan. "Container Escape Detection for Edge Devices". In: *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. Coimbra Portugal: ACM, Nov. 2021, pp. 532–536. ISBN: 978-1-4503-9097-2. DOI: 10.1145/3485730.3494114. (Visited on 12/11/2022).

[35] Yulong Wang, Qixu Wang, Xingshu Chen, Dajiang Chen, Xiaojie Fang, Mingyong Yin, and Ning Zhang. "ContainerGuard: A Real-Time Attack Detection System in Container-Based Big Data Platform". In: *IEEE Transactions on Industrial Informatics* 18.5 (May 2022), pp. 3327–3336. ISSN: 1941-0050. DOI: 10.1109/TII.2020.3047416.

[36] S. Chakravarthi, R. Kannan, V. Natarajan, and Xiao-Zhi Gao. "Deep Learning Based Intrusion Detection in Cloud Services for Resilience Management". In: *Computers, Materials & Continua* 71.3 (2022), pp. 5117–5133. ISSN: 1546-2218, 1546-2218. DOI: 10.32604/cmc.2022.022351. (Visited on 12/11/2022).

[37] Jingfei Shen, Fanping Zeng, Weikang Zhang, Yufan Tao, and Shengkun Tao. "A Clustered Learning Framework for Host Based Intrusion Detection in Container Environment". In: *2022 IEEE International Conference on Communications Workshops (ICC Workshops)*. Seoul, Korea, Republic of: IEEE, May 2022, pp. 409–414. ISBN: 978-1-66542-671-8. DOI: 10.1109/ICCWorkshops53468.2022.9814620. (Visited on 12/11/2022).

[38] Martin Grimmer, Martin Max Röhling, D Kreusel, and Simon Ganz. "A Modern and Sophisticated Host Based Intrusion Detection Data Set". In: *IT-Sicherheit als Voraussetzung für eine erfolgreiche Digitalisierung* (2019), pp. 135–145.

[39] Yulong Wang, Qixu Wang, Xue Qin, Xingshu Chen, Bangzhou Xin, and Run Yang. "DockerWatch: A Two-Phase Hybrid Detection of Malware Using Various Static Features in Container Cloud". In: *Soft Computing* (Oct. 2022). ISSN: 1432-7643, 1433-7479. DOI: 10.1007/s00500-022-07546-2. (Visited on 12/07/2022).

[40] Yuhang Lin, Olufogorehan Tunde-Onadele, Xiaohui Gu, Jingzhu He, and Hugo Latapie. "SHIL: Self-Supervised Hybrid Learning for Security Attack Detection in Containerized Applications". In: *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. Sept. 2022, pp. 41–50. DOI: 10.1109/ACSOS55765.2022.00022.

[41] Yuhang Lin, Olufogorehan Tunde-Onadele, and Xiaohui Gu. "CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications". In: *Annual Computer Security Applications Conference*. ACSAC '20. New York, NY, USA: Association for Computing Machinery, Dec. 2020, pp. 179–188. ISBN: 978-1-4503-8858-0. DOI: 10.1145/3427228.3427236. (Visited on 12/10/2022).

[42] Tin Kam Ho. "Random Decision Forests". In: *Proceedings of the Third International Conference on Document Analysis and Recognition (Volume 1) - Volume 1*. ICDAR '95. USA: IEEE Computer Society, Aug. 1995, p. 278. ISBN: 978-0-8186-7128-9. (Visited on 01/14/2023).

[43] Holger Gantikow, Tom Zohner, and Christoph Reich. "Container Anomaly Detection Using Neural Networks Analyzing System Calls". In: *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Västerås, Sweden: IEEE, Mar. 2020, pp. 408–412. ISBN: 978-1-72816-582-0. DOI: 10.1109/PDP50117.2020.00069. (Visited on 12/07/2022).

[44] Yulong Wang, Xingshu Chen, Qixu Wang, Run Yang, and Bangzhou Xin. "Unsupervised Anomaly Detection for Container Cloud Via BILSTM-Based Variational Auto-Encoder". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2022, pp. 3024–3028. DOI: 10.1109/ICASSP43922.2022.9747341.

[45] Gabriel R. Castanhel, Tiago Heinrich, Fabrício Ceschin, and Carlos Maziero. "Taking a Peek: An Evaluation of Anomaly Detection Using System Calls for Containers". In: *2021 IEEE Symposium on Computers and Communications (ISCC)*. Sept. 2021, pp. 1–6. DOI: 10.1109/ISCC53001.2021.9631251.

[46] Pinchen Cui and David Umphress. "Towards Unsupervised Introspection of Containerized Application". In: *2020 the 10th International Conference on Communication and Network Security*. ICCNS 2020. New York, NY, USA: Association for Computing Machinery, Mar. 2021, pp. 42–51. ISBN: 978-1-4503-8903-7. DOI: 10.1145/3442520.3442530. (Visited on 12/12/2022).

[47] Olufogorehan Tunde-Onadele, Jingzhu He, Ting Dai, and Xiaohui Gu. "A Study on Container Vulnerability Exploit Detection". In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. June 2019, pp. 121–127. DOI: 10.1109/IC2E.2019.00026.

[48] Qingfeng Du, Tiandi Xie, and Yu He. "Anomaly Detection and Diagnosis for Container-based Microservices with Performance Monitoring". In: *International Conference on Algorithms and Architectures for Parallel Processing* (2018).

[49] Supriya Kamthania. "A Novel Deep Learning RBM Based Algorithm for Securing Containers". In: *2019 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*. Nov. 2019, pp. 1–7. DOI: 10.1109/WIECON-ECE48653.2019.9019985.

[50] Chin-Wei Tien, Tse-Yung Huang, Chia-Wei Tien, Ting-Chun Huang, and Sy-Yen Kuo. "KubAnomaly: Anomaly Detection for the Docker Orchestration Platform with Neural Network Approaches". In: *Engineering Reports* 1.5 (2019), e12080. ISSN: 2577-8196. DOI: 10.1002/eng2.12080. (Visited on 05/08/2022).

[51] Mubin Ul Haque, M. Mehdi Kholoosi, and M. Ali Babar. "KGSecConfig: A Knowledge Graph Based Approach for Secured Container Orchestrator Configuration". In: *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. Honolulu, HI, USA: IEEE, Mar. 2022, pp. 420–431. ISBN: 978-1-66543-786-8. DOI: 10.1109/SANER53432.2022.00057. (Visited on 01/09/2023).

[52] Joanna Kosinska and Maciej Tobiasz. "Detection of Cluster Anomalies With ML Techniques". In: *IEEE Access* 10 (2022), pp. 110742–110753. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2022.3216080. (Visited on 12/30/2022).

[53] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

[54] Qiqing Deng, Xinrui Tan, Jing Yang, Chao Zheng, Liming Wang, and Zhen Xu. "A Secure Container Placement Strategy Using Deep Reinforcement Learning in Cloud". In: *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. Hangzhou, China: IEEE, May 2022, pp. 1299–1304. ISBN: 978-1-66540-527-0. DOI: 10.1109/CSCWD54268.2022.9776226. (Visited on 12/12/2022).

[55] Thanh Thi Nguyen and Vijay Janapa Reddi. "Deep Reinforcement Learning for Cyber Security". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–17. ISSN: 2162-2388. DOI: 10.1109/TNNLS.2021.3121870.

[56] Huanruo Li, Yunfei Guo, Penghao Sun, Yawen Wang, and Shumin Huo. "An Optimal Defensive Deception Framework for the Container-based Cloud with Deep Reinforcement Learning". In: *IET Information Security* 16.3 (May 2022), pp. 178–192. ISSN: 1751-8709, 1751-8717. DOI: 10.1049/ise2.12050. (Visited on 12/11/2022).

[57] Tong Kong, Liming Wang, Duohe Ma, Zhen Xu, Qian Yang, and Kai Chen. "A Secure Container Deployment Strategy by Genetic Algorithm to Defend against Co-Resident Attacks in Cloud Computing". In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. Zhangjiajie, China: IEEE, Aug. 2019, pp. 1825–1832. ISBN: 978-1-72812-058-4. DOI: 10.1109/HPCC/SmartCity/DSS.2019.00251. (Visited on 12/14/2022).

AUTHORS

Ilter Taha Aktolga received his B.S. degree in computer engineering at Middle East Technical University (METU) in 2021. He is currently a graduate student pursuing a master's degree at METU. He has gained research and industry experience through various positions, including working as an undergraduate researcher at KOVAN Robotics Research Laboratory at METU with a TUBITAK scholarship from June 2019 to June 2020 and as a cloud developer at Arcelik Global from May 2021 to August 2021. Currently, he is working as a software engineer at ASELSAN. His research interests include the fields of machine learning, container security, and backend development with an emphasis on microservices.

Elif Sena Kuru is an undergraduate student pursuing her bachelors degree in computer engineering at Middle East Technical University (METU) since 2019. Her research interests include cloud security and machine learning.

Yigit Sever received a B.S. degree in computer engineering at TED University, Turkey, in 2016, and a M.S. degree in computer engineering at Hacettepe University, Turkey, in 2019. He is currently a Ph.D. candidate in computer engineering at Middle East Technical University (METU), Turkey, where he is also working as a research assistant since 2020. His research interests include cloud security with a strong focus on container security, user and Internet privacy and distributed systems. He is a member of Wireless Systems, Networks and Cybersecurity Laboratory, METU.

Pelin Angin (Member, IEEE) received a B.S. degree in computer engineering at Bilkent University, in 2007, and a Ph.D. degree in computer science from Purdue University, USA, in 2013. From 2014 to 2016, she worked as a visiting assistant professor and a postdoctoral researcher at Purdue University. She is currently an associate professor in computer engineering at Middle East Technical University. Her research interests include the fields of cloud computing, the IoT security, distributed systems, 5G networks, data mining, and blockchain. She is among the founding members of the Systems Security Research Laboratory and an affiliate of the Wireless Systems, Networks and Cybersecurity Laboratory, METU.