SHIL: Self-Supervised Hybrid Learning for Security Attack Detection in Containerized Applications

Yuhang Lin
North Carolina State University
ylin34@ncsu.edu

Olufogorehan Tunde-Onadele
North Carolina State University
oatundeo@ncsu.edu

Xiaohui Gu
North Carolina State University
xgu@ncsu.edu

Jingzhu He
ShanghaiTech University
hejzh1@shanghaitech.edu.cn

Hugo Latapie
Cisco
hlatapie@cisco.com

***Abstract*—Container security has received much research attention recently. Previous work has proposed to apply various machine learning techniques to detect security attacks in containerized applications. On one hand, supervised machine learning schemes require sufficient labelled training data to achieve good attack detection accuracy. On the other hand, unsupervised machine learning methods are more practical by avoiding training data labelling requirements, but they often suffer from high false alarm rates. In this paper, we present SHIL, a self-supervised hybrid learning solution, which combines unsupervised and supervised learning methods to achieve high accuracy without requiring any manual data labelling. We have implemented a prototype of SHIL and conducted experiments over 41 real world security attacks in 28 commonly used server applications. Our experimental results show that SHIL can reduce false alarms by 39-91% compared to existing supervised or unsupervised machine learning schemes while achieving a higher or similar detection rate.**

***Index Terms*—Container Security, Security Attack Detection, Hybrid Machine Learning**

I. Introduction

Container-based distributed system platforms have gained tremendous popularity in many real world applications because of their cost-effectiveness and accessibility. However, containers also open new attack surfaces to malicious attackers. Recent studies [1], [2], [3] have shown that containers are vulnerable to various security attacks. For example, Tesla suffered a cryptojacking attack in February 2018 [4]. The hackers infiltrated Tesla's Kubernetes console to steal sensitive data such as telemetry. Besides data exposure, the hackers performed crypto mining with low resource intensity to evade detection. Furthermore, Apache Log4j recently disclosed a vulnerability in December 2021 that seriously affected distributed systems worldwide, including container clusters [5]. Java software systems extensively use the open-source Log4j logging framework, exposing them to remote code execution attacks. The Google Cloud open source insights team estimated that over 35,000 artifacts in the Maven Central repository are vulnerable, four times that of the average vulnerability [6]. In just four days after the disclosure, Check Point Research reported over 800,000 global attacks to the vulnerability [7].

Traditional intrusion detection systems [8] typically employ rule-based approaches, which however cannot adapt to highly dynamic container environments and often miss detecting emergent attack behaviors that have not been captured by existing detection rules. Previous work [9], [10], [11], [12] has proposed to apply different machine learning methods including supervised learning, unsupervised learning, and semi-supervised learning, to achieve effective security attack detection. Supervised learning typically achieves higher detection accuracy than unsupervised learning. However, it is difficult to collect sufficient high quality labelled training data in highly ephemeral container-based computing environments. Anomaly detection methods based on unsupervised learning are much easier to be deployed in real world dynamic computing environments, which do not require any labelled training data. However, due to the lack of labelled training data, anomaly detection methods often suffer from high false alarms. Previous work also proposed semi-supervised learning methods [12] for security attack detection, which start with a trained supervised learning model and use unsupervised methods to augment the supervised model by providing labels to unlabelled data. However, the semi-supervised approach still requires labelled training data in order to train the initial supervised model.

In this paper, we present a new *self-supervised hybrid learning* (SHIL) system for performing adaptive online security attack detection in dynamic containerized applications. SHIL aims at improving security detection accuracy without requiring any labelled training data which are particularly difficult to obtain in ephemeral container-based systems. In contrast to semi-supervised learning, which starts from supervised models, SHIL starts with unsupervised models and employs supervised learning methods for cross validation purposes only. The rationale behind our approach is based on the observation that most false alarms occur when the measurement sample is within close vicinity of the anomaly detection threshold, which is called a *boundary case*. Cross-validations using multiple different learning methods over boundary cases can effectively filter out false alarms without missing true attack anomalies.

SHIL leverages non-intrusive, light-weight system call mon-

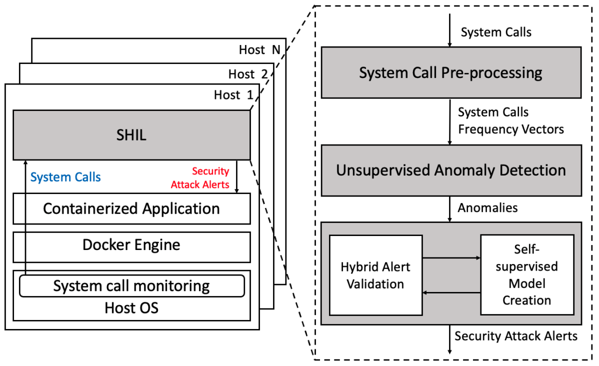


Fig. 1: System overview of SHIL.

itoring tools [13] to achieve practical security attack detection for containers. SHIL consists of three key modules: 1) *unsupervised anomaly detection*, which leverages autoencoder neural networks [14] to achieve fast attack detection without requiring any labelled training data; 2) *hybrid alert validation*, which identifies boundary case anomalies and performs false alarm filtering by cross validating the anomalies using the supervised learning method random forest tree [15]; and 3) *self-supervised model creation*, which performs outlier detection using isolation forest [16] over a window of recent system call data upon an attack alert raised by either SHIL or a third party attack detection tool. The outlier detection allows SHIL to generate training labels automatically to achieve self-supervised learning.

Specifically, this paper makes the following contributions:

- We propose a new self-supervised hybrid machine learning approach to achieving effective security attack detection with few false alarms.
- We describe a self-supervised model creation method which leverages both the anomalies produced by SHIL and an outlier detection method to produce training labels automatically for the supervised learning method.
- We implement and evaluate a prototype of SHIL over 41 recent real critical vulnerabilities with high CVSS scores including the high impact Apache Log4j vulnerability, in 28 commonly used server applications.

Our experimental results show that SHIL can reduce false alarms by 39% to 91% compared to existing unsupervised or supervised learning methods while achieving higher or similar detection rates. SHIL is light-weight, which makes it practical for large-scale container based computing environments.

The rest of the paper is structured as follows. Section II presents the system design in detail. Section III describes our experimental evaluation methodology and our experimental results. Section IV compares our work with related work. Section V concludes this paper.

II. System Design

In this section, we present the design of the SHIL system. We first provide a system overview. Next, we describe each component in detail.

SHIL takes a self-supervised hybrid machine learning approach to security attack detection. As highlighted in Figure 1, the system consists of four key integrated components: 1) *system call pre-processing*, 2) *unsupervised anomaly detection*, 3) *hybrid alert validation*, and 4) *self-supervised model creation*.

SHIL leverages light-weight system call monitoring tool [13] to detect security attacks. Previous intrusion detection work have revealed that the program executes a locally consistent set of system call sequences during normal operations and attacks often exhibit significant changes in system call invocations [17], [18]. The system call pre-processing module extracts a *system call frequency vector* feature from raw system call traces. Each system call frequency vector denotes the number of invocations for all system call types (e.g., `sys_read`) within the sampling interval (e.g., 100 milliseconds). Table I shows examples of frequency vectors from an OpenSSH 7.2p2 container during an exploit of the CVE-2016-6515 vulnerability. The attack starts at timestamp 1586496672891 and ends at timestamp 1586496674291. During the attack, we observe the frequency of the *execve* and *lstat* system calls increase when the attack starts being detected until the attack completes. Meanwhile, the frequencies of both *access* and *mmap* calls sharply increase during the same period. Notice that our self-supervised hybrid learning approach can be applied to any system call features (e.g., n-grams [17]) adopted by the intrusion detection system. In this paper, we choose to use the system call frequency vector features for both low cost training and real-time attack detection.

The unsupervised anomaly detection component detects anomalies in the system call frequency vector data using an autoencoder neural network. The trained model computes the difference between an input vector and its reconstruction of the vector into a value called reconstruction error. The model compares the reconstruction error against a pre-defined percentile value (e.g., 99 percentile value of all reconstruction errors) to detect anomalies. Section II-A will provide details about this module.

The hybrid alert validation component checks whether the detected anomaly is a boundary case (e.g., within a small deviation from the normal value) and invokes the supervised model to perform cross validation if it is considered to be a boundary case. The goal is to filter out most false alarms produced by the boundary cases and only raise attack alerts when both unsupervised and supervised models confirm the alert. Section II-B will provide details about this module.

To achieve supervised learning without requiring manual data labelling, SHIL adopts a self-supervised learning approach using the self-supervised model creation method. Upon an attack alert, SHIL creates a supervised attack detection model such as random forest using a window of system call frequency vector data before and after the attack is detected. Instead of relying on manual data labelling, SHIL automatically creates training data labels by performing outlier detection using isolation forest models [16]. Section II-C will provide details about this module.

TABLE I: A frequency vector sample for the *OpenSSH* application (CVE-2016-6515). An attack is triggered at t = 1586496672891. Anomaly detection raises alarms from t = 1586496673091 to t = 1586496673191. The entries with asterisks (*) are detected outliers.

Timestamp	**System Call Frequency**			
	access	**execve**	**lstat**	**mmap**
1586496672491	0	0	0	0
1586496672591	0	0	0	0
1586496672691	0	0	0	0
1586496672791	0	0	0	0
1586496672891 (attack starts)	*0*	*0*	*0*	*0*
1586496672991	79	7	0	155
1586496673091 (attack detected)	**136**	**41**	**12**	**420**
1586496673191*	**268**	**51**	**16**	**702**
1586496673291*	209	46	24	637
1586496673391	164	23	8	422
1586496673491	70	32	24	290
1586496673591	190	21	12	454
1586496673691	76	12	12	297
1586496673791*	129	55	28	386
1586496673891	130	5	0	353
1586496673991	82	20	4	249
1586496674091*	159	42	20	439
1586496674191	79	3	8	260
1586496674291 (attack succeeds)	91	53	16	250
1586496674391	192	12	0	468
1586496674491	50	1	0	142
1586496674591	0	0	0	0
1586496674691	0	0	0	0

A. Unsupervised Anomaly Detection

In contrast to previous semi-supervised learning methods which start from supervised models, SHIL starts from unsupervised anomaly detection models. We choose our design based on two key rationales: 1) unsupervised anomaly detection does not require labelled training data as it relies on recognizing deviations from normal behaviors; and 2) unsupervised anomaly detection has the ability to detect unknown attacks. So SHIL can inherit all the advantages of the unsupervised learning methods by only involving supervised models for performing cross validations on boundary cases.

Our unsupervised anomaly detection leverages autoencoder neural networks. The autoencoder neural network is an artificial neural network model with a symmetric structure, capable of reconstructing its input as the output. The network consists of two major sections: the encoder and the decoder. The encoder reduces the frequency vectors into increasingly lower dimensions from the input layer to the narrowest hidden layer at the autoencoder center. Conversely, the decoder reconstructs the low dimension vector representation from the hidden layer to the output layer. We train each autoencoder repeatedly with certain number of iterations (e.g., 10 iterations) to allow it to rapidly adjust its weights under various system call activity.

During detection, the difference between the input and output of the autoencoder model is referred to as the reconstruction error. Anomalous frequency vector samples that deviate from the model of normal samples are likely to be reconstructed poorly, resulting in more substantial error degrees than others. Therefore, we implement the anomaly detection by comparing the reconstruction error of the current sample with a pre-defined reconstruction error threshold. Specifically, we select a certain percentile value (e.g. 99 percentile) of the reconstruction errors during the training phase as the threshold of our anomaly detection model. The rationale is that the majority of the training data are normal data which have small reconstruction errors.

We use a model ensemble approach that trains a separate model for each group of containers that run the same application with the same version. For example, we create an anomaly detection model to monitor a group of containers running *JBoss 6.1.0*. The application specific models provide higher accuracy over monolithic models trained over different applications because of fewer conflicting training data [10].

B. Hybrid Alert Validation

The anomaly detection percentile threshold typically controls the trade-off between the detection rate and false positive rate. If we want the anomaly detection model detects more attacks, we need to configure relatively lower percentile threshold in order to capture more anomalous behaviors. However, lower percentile threshold inevitably produces more false alarms since the anomaly detection model is more likely to detect rogue anomalies caused by dynamic container-based computing environments and transient workload fluctuations. During our experiments, we observe that majority of those false alarms occur in boundary cases where the reconstruction error is above the anomaly detection threshold within a small range. For example, the error range between 100% and 110% of the threshold contains over 50% of the false alarms. Thus, we propose to identify those boundary cases and perform cross-validations over those boundary cases using a self-supervised model trained by a supervised learning method.

Recall that the unsupervised anomaly detection calculates the reconstruction error of each frequency vector that is continuously compared against an error threshold to make an anomaly decision. We define the error range above the threshold that contains the majority of false alarms as the *boundary case*. SHIL then feeds the boundary case anomaly into a pre-trained supervised model for cross validation. If the supervised model does not classify the measurement sample as anomalous, SHIL deems it to be a false alarm from the unsupervised anomaly detection model and drops the alert. Notice that SHIL only applies the cross validation over the boundary cases so that SHIL can avoid dropping alerts about unknown attacks that are often missed by supervised models.

C. Self-supervised Model Creation

Supervised models can typically achieve high detection accuracy when sufficient high quality labelled training data are available. However, it is often difficult to obtain labelled training data in production environments, especially in highly dynamic container-based computing environments. Moreover, for dynamic advanced attacks, it is extremely challenging if not totally impossible to accurately label each measurement sample as normal or abnormal at a fine-grained time scale

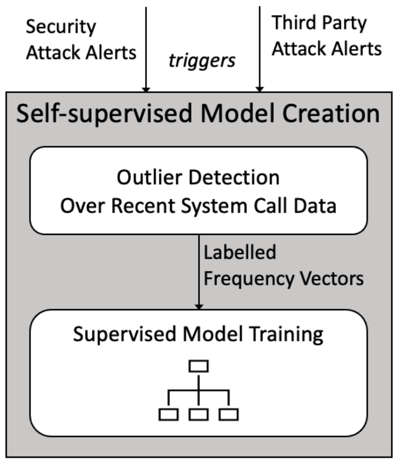

Security Attack Alerts
triggers
Third Party Attack Alerts
Self-supervised Model Creation
Outlier Detection Over Recent System Call Data
Labelled Frequency Vectors
Supervised Model Training

Fig. 2: Self-supervised model creation.

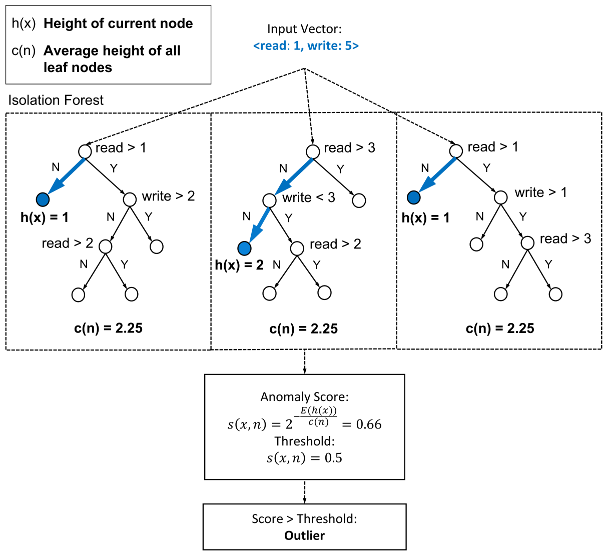


Fig. 3: An example of outlier detection with the isolation forest model. The leaf nodes with the shortest heights from their root nodes tend to be outliers.

(e.g., every 10 milliseconds) required by supervised learning models. When an attack is first started, it may not manifest in the system call frequency vector changes immediately. For example, in Table I, the attack is triggered at timestamp 1586496672891, however there is not much change until timestamp 1586496672991. After that we can see the increase of `access`, `execve`, `lstat` and `mmap` system call frequencies. In addition, the attack may have a lasting impact resulting in non-deterministic behaviors (i.e., normal, abnormal, or mixed activities) after the attack succeeds.

To address the challenges of lacking high-quality labelled training data, we propose a self-supervised model training approach to achieving automatic data labelling, as shown in Figure 2. Intuitively, measurement samples outside the attack period represent normal fluctuating behaviors in a dynamic application and measurement samples collected during the attack period represent abnormal behaviors incurred by the attack. However, advanced attacks might not exhibit abnormal behaviors constantly throughout the attack period as shown in our experiments. If we label all measurement samples during the attack period as anomaly data, it is highly possible to create a model that tends to raise many false alarms. To address the challenge, we introduce an outlier detection process to preprocess the training data.

Specifically, our self-supervised model training consists of the following major steps. First, we perform outlier detection over all measurement data during the attack period identified by our unsupervised learning methods or other third-party attack detection tools. Second, we perform a similarity check between each outlier detected during the attack period with all the normal measurement samples collected during a small window preceding the attack detection time. In our experiments, we employed Manhattan distance between two frequency vectors to measure the similarity. We then further filter those outliers which resemble the normal execution behavior. The rationale behind our approach is to capture the true attack behavior while minimizing the false positive likelihood. Thus, We only label those true outliers as anomalous in the supervised model training. Those filtered outliers which resemble normal execution data will be relabelled into "normal" to reinforce the normal behavior training.

During our experimental study, we observe that only a small portion of samples during the attack period contain significant changes in system call frequencies, which are often detected as outliers. Thus, our approach provides fine-grained labelling instead of assuming all measurement samples during the attack period as abnormal. Our experiments show that such a fine-grained data labeling approach can effectively induce high quality supervised learning models.

We employ the isolation forest [16] outlier detection method to generate fine-grained labels. The isolation forest consists of an ensemble of decision trees. The goal of each decision tree is to isolate each input vector from the others. Each tree is split on a random value in the possible range of a randomly selected system call type (e.g., read) until all vectors are separated. Since outliers are uncommon and numerically different from normal samples, it takes fewer decisions to distinguish them from others. Thus, outliers are found closer to the root of the isolation trees.

Figure 3 illustrates the outlier detection of a simple frequency vector sample on a constructed isolation forest. The input vector is isolated in each tree by following the decision tree path consisting of the highlighted links. For instance, in the left-most tree, the vector is separated with a single split of the $read > 1$ node. Whereas, the tree in the center of the forest separates the vector with two decision nodes: $read > 3$ and $write < 3$.

To determine whether the vector is an outlier, an anomaly score is computed as follows.

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \quad (1)$$

Given a set of n instances, the anomaly score $s(x, n)$ of an instance x in the isolation forest is defined by Equation (1), where $h(x)$ is the path length of x from the root, $E(h(x))$ is the average of $h(x)$ from the collection of isolation trees and $c(n)$ is the average path length of unsuccessful search in an isolation tree. In a binary search tree, $c(n)$ is estimated by the average height from the root to its leaf nodes [16]. When s is close to 1, it is nearly certain that x is an outlier. Whereas, when s is close to 0, x is highly likely to be a normal sample. In Figure 3, the heights $h(x)$ of the vector sample in each tree from left to right are one, two, and one, respectively, while the average height of the leaf nodes $c(n)$ is 2.25. The sample generates an anomaly score of 0.66. If we set the outlier detection threshold as 0.5, this input vector is labelled as an outlier.

For example, Table I shows the labelling results produced by our outlier detection method. Those entries marked with asterisk (*) indicate detected outliers which will be labelled as anomaly in the supervised model's training data. The measurement samples before the attack detection time and all non-outlier measurements are labelled as normal training data. We can see that only those outliers within the attack period show abnormal behavior while those non-outlier samples still exhibit similar behaviors to those measurements during the normal execution period.

After the outlier detection module produces labelled training data, it feeds the labelled training data into a supervised learning model. During our experiments, we choose the random forest (RF) learning method as our supervised learning method [15] for its simplicity and effectiveness. The RF model is an ensemble of many decision trees, which makes its final classification based on the majority classification of its constituents. During training, each tree chooses a split value from a random subset of its features to optimize its anomaly classification decision. These characteristics make the RF model resilient to noises. For a single tree, the fraction of abnormal samples in the output leaf node yields a probability value. The overall prediction probability is the the average prediction probability over all the decision trees. When the prediction probability is above a pre-defined threshold such as 60%, the frequency vector is classified as abnormal.

III. Experimental Evaluation

In this section, we first describe our evaluation methodology. Next, we compare SHIL with a set of alternative schemes. We implement a prototype of SHIL and evaluate it on a desktop with four 3.4 GHz cores and 8 GB memory running Ubuntu 18.04 64-bit.

A. Evaluation Methodology

Real-world vulnerabilities. We evaluate 41 vulnerabilities from 28 applications listed in the common vulnerabilities and exposures (CVE) database, which include many applications commonly used in production environments [19], [20]. Table II shows the complete list of the CVEs, including the common vulnerability scoring system (CVSS) score v2.0, application name, and version of each entry. We focus on CVEs in recent years, including the recent high-impacting Log4j CVE (CVE-2021-44228). We classify the vulnerabilities into six categories according to the threat impact of the attack. The threat impact categories comprise attacks that 1) return a shell and execute arbitrary code, 2) execute arbitrary code, 3) disclose credential information, 4) consume excessive CPU, 5) crash the application, and 6) escalate privilege level.

TABLE II: List of explored real-world vulnerabilities.

Threat Impact	CVE ID	CVSS Score	Application	Version
Return a shell and execute arbitrary code	CVE-2012-1823	7.5	PHP	5.4.1
	CVE-2014-3120	6.8	Elasticsearch	1.1.1
	CVE-2015-1427	7.5	Elasticsearch	1.4.2
	CVE-2015-2208	7.5	phpMoAdmin	1.1.2
	CVE-2015-3306	10.0	ProFTPd	1.3.5
	CVE-2015-8103	7.5	JBoss	6.1.0
	CVE-2016-3088	7.5	Apache ActiveMQ	5.11.1
	CVE-2016-9920	6.0	Roundcube	1.2.2
	CVE-2016-10033	7.5	PHPMailer	5.2.16
	CVE-2017-7494	10.0	Samba	4.5.9
	CVE-2017-8291	6.8	Ghostscript	9.2.1
	CVE-2017-11610	9.0	Supervisor	3.3.2
	CVE-2017-12149	7.5	JBoss	6.1.0
	CVE-2017-12615	6.8	Apache Tomcat	8.5.19
Execute arbitrary code	CVE-2014-6271	10.0	Bash	4.2.37
	CVE-2015-8562	7.5	Joomla	3.4.2
	CVE-2016-3714	10.0	ImageMagick	6.7.9
	CVE-2017-5638	10.0	Apache Struts 2	2.5.0
	CVE-2017-12794	4.3	Django	1.11.4
	CVE-2018-11776	9.3	Apache Struts 2	2.3.34
	CVE-2018-16509	9.3	Ghostscript	9.23
	CVE-2018-19475	6.8	Ghostscript	9.25
	CVE-2019-6116	6.8	Ghostscript	9.26
	CVE-2019-5420	7.5	Rails	5.2.2
	CVE-2020-17530	7.5	Apache Struts 2	2.5.25
	CVE-2021-44228 (log4j)	9.3	Apache Solr	8.11.0
Disclose credential information	CVE-2014-0160	5.0	OpenSSL	1.0.1e
	CVE-2015-5531	5.0	Elasticsearch	1.6.0
	CVE-2017-7529	5.0	Nginx	1.13.2-1
	CVE-2017-8917	7.5	Joomla	3.7.0
	CVE-2018-15473	5.0	OpenSSH	7.7p1
	CVE-2020-1938	7.5	Apache Tomcat	9.0.30
	CVE-2021-28164	5.0	Jetty	9.4.37
	CVE-2021-28169	5.0	Jetty	9.4.40
	CVE-2021-34429	5.0	Jetty	9.4.40
	CVE-2021-41773	4.3	Apache HTTP Server	2.4.49
Consume excessive CPU	CVE-2014-0050	7.5	Apache Commons FileUpload	1.3.1
	CVE-2016-6515	7.8	OpenSSH	7.2p2
Crash the application	CVE-2015-5477	7.8	BIND	9
	CVE-2016-7434	5.0	NTP	1.4.2.8
Escalate privilege level	CVE-2017-12635	10.0	CouchDB	2.1.0

Experiment Setup. For each vulnerability, we set up four Docker containers under Ubuntu 16.04 LTS. We use Apache JMeter to deliver a different multiple of workload to each of the four containers, i.e. 1x, 2x, 4x and 8x. We design suitable workload for each vulnerability according to the kind of traffic the main containerized application accepts. For example, we simulate traffic using HTTP GET requests to the containers of CVE-2020-1938 because the application, Apache Tomcat, is a web server software. Once the container starts running, we use Sysdig to collect seven minutes of its system call activity, unless the container exits early due to an attack (such as the attack to CVE-2015-5477 that crashes the BIND application). The short duration gives enough samples for our models and aligns with the ephemeral nature of containers.

Each experiment is conducted as follows. First, we let the container run for four minutes under the appropriate normal workload. Next, we trigger the attack using open source exploit code around the start of the fifth minute and let the attack continue running until it succeeds. Meanwhile, the exploit program logs the attack triggering time and attack success time due to our modification of the original program from exploit databases. Lastly, we stop the attack where applicable. After finishing the experiment, we process the collected system calls into system call frequency vectors using a sampling rate of 100 milliseconds.

SHIL Prototype Implementation. We use TensorFlow to build an autoencoder (AE) model using four hidden layers with 278 neurons in the first and fourth hidden layers and 70 neurons in the second and third hidden layers. For our autoencoder model, we measure the reconstruction error using root mean square error (RMSE) as defined in Equation (2), where N is the number of samples, y_i is the value of input, and $\hat{y}_i$ is the value of the output.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}} \tag{2}$$

We adopt the scikit-learn implementation of isolation forest and set the contamination threshold to be 0.5 after experimentation. Each container has its own attack and workload characteristics, thus each container has its own isolation forest model that fits and predicts the outliers within its alerted attack period.

The outliers detected may contain some data points similar to the normal period. We then calculate the pairwise Manhattan distances between outliers and normal data points. If the smallest Manhattan distance between an outlier and a normal data point is less than the similarity threshold, we remove this outlier. After several experiments, we choose 5 as the similarity threshold.

We apply the scikit-learn implementation of random forest to build our supervised model. Specifically, we use 100 decision trees with no specified maximum depth. We observe that adding more decision trees does not improve the precision but consumes more CPU and memory resources.

Alternative approaches. To evaluate the efficacy of SHIL, we compare SHIL with several alternative real-time, light-weight security attack detection methods [10], [11] that have been proposed for container systems. We also implement pure-supervised or pure-unsupervised learning methods for evaluating the efficacy of our hybrid learning approaches.

- **Classified Distributed Learning (CDL) [10]**: We run CDL using 95 and 99 percentile anomaly detection thresholds. CDL uses the same autoencoder model as used in SHIL.
- **Self-patch [11]**: To make a fair comparison, we only compare our work with the attack detection part of the self-patch. Self-patch uses an autoencoder model with four hidden layers to detect attacks to containers. There are 256 neurons in the first and fourth hidden layers, and 128 neurons in the second and third hidden layers. Self-patch applies 99 percentile anomaly detection threshold only.
- **Supervised (RF)**: We use a pure supervised random forest model. The hyperparameters used in the random forest model are the same as the random forest model used in SHIL.
- **Supervised (CNN)**: We use the convolution neural network (CNN) learning method, which has been recently applied to cybersecurity [21]. We use Keras, with Tensorflow as the backend, to implement this model. This model consists of two 1-D convolution layers with the rectified linear activation function (ReLU) and a kernel size of 5. The reason for choosing 1-D convolution layer is that the layer moves along one dimension, which makes it applicable for time-series data. The kernel size represents how many features are considered every time the kernel moves across a vector sample. The first CNN layer has eight filters while the second layer has four filters. The CNN then includes a flatten layer to flatten the output from the second convolution layer. Finally, the network has a dense layer of sigmoid activation function neurons to predict the probability of the input frequency being abnormal. Our CNN model is trained for 35 iterations using the Adam optimizer with a learning rate of 0.001. To train a more robust model, we shuffle the training set during training.

Evaluation metrics. We define *DC* to be the number of containers that are under attack and correctly identified by the detector, and *MC* to be the number of containers that are under attack and incorrectly missed by the detector. We use false positive (FP) to denote the number of measurement samples the detector falsely identifies, and true negative (TN) to be the number of samples the detector correctly rejects. Using the above definitions, the detection rate and false positive rate (FPR) are given by equations (3) and (4), respectively.

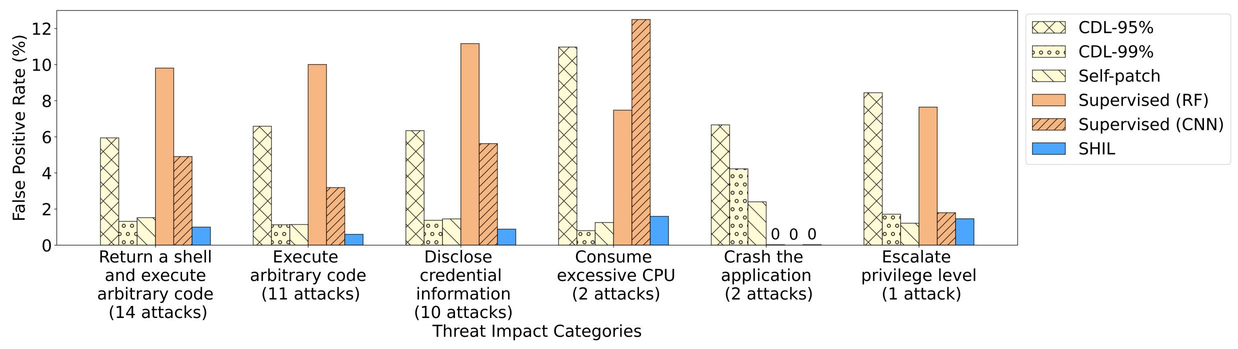
$$detection\ rate = \frac{DC}{DC + MC} \tag{3}$$

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

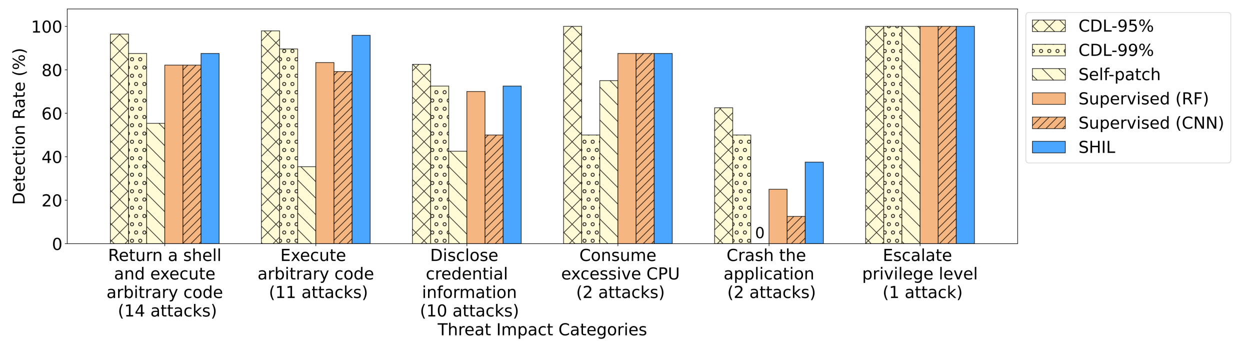
We introduce *lead time* to denote the duration from the time the first alert is raised by the detection system to the time the attack is successful if no action is triggered before then.

B. Results Analysis

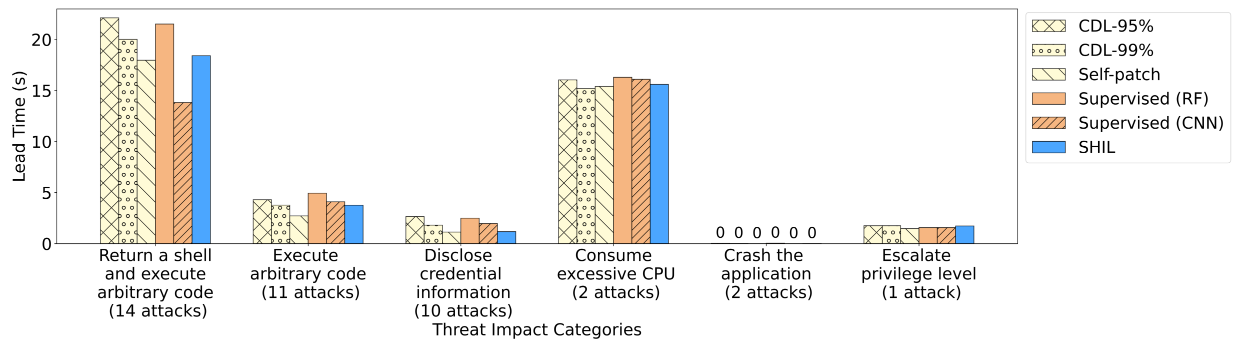
In this section, we present the analysis of the experiment results. Table III compares the detection results among CDL [10] using 95 percentile (CDL-95%) and 99 percentile (CDL-99%) anomaly detection thresholds, Self-patch [11], the pure supervised RF and CNN models, and our SHIL approach



(a) False positive rate comparison.



(b) Detection rate comparison.



(c) Lead time comparison.

Fig. 4: Model comparisons among unsupervised autoencoder using 95% and 99% thresholds, supervised RF, supervised CNN, combined, and SHIL.

TABLE III: Comparison with alternative approaches.

Model	Detection rate	FPR	Lead time
CDL-95%	92.07%	6.57%	10.29s
CDL-99%	81.10%	1.40%	9.17s
Self-patch	45.73%	1.41%	7.99s
Supervised (RF)	77.44%	9.55%	10.24s
Supervised (CNN)	70.73%	4.63%	7.22s
SHIL 120% Boundary	**89.02%**	**2.04%**	**9.51s**
SHIL 200% Boundary	**84.15%**	**0.85%**	**8.48s**

using 120% (SHIL-120%) and 200% boundary case threshold (SHIL-200%).

For CDL, increasing the anomaly detection threshold from 95 percentile to 99 percentile reduces the false positive rate from 6.57% to 1.40%. However, we also see a big drop in detection rate from from 92.07% to 81.10%. Self-patch has a similar FPR as CDL-99%, but its detection rate is only 45.73%. In contrast, SHIL-120% model can reduce false positive rate significantly by 69.0% while maintaining similar detection rate and lead time compared to pure unsupervised model CDL-95%. By increasing the boundary cases to a higher threshold, SHIL-200% can achieve a higher detection rate while reducing the false positive rate by 39.3%, compared to the pure unsupervised model CDL-99%. By having a self-supervised model to effectively filter the false positives, SHIL can adopt an unsupervised model with lower anomaly detection thresholds, thus achieving a higher detection rate and lower false positve rate at the same time.

The pure supervised models perform poorly which suffers from both low detection rate and high false alarms due to low quality labelled training data [22]. When comparing with the pure supervised RF and CNN models, SHIL-200% can reduce the false positive rate by 91.1% and 81.6%, respectively with higher detection rate and similar lead time.

Figure 4a shows the false positive rate of different models under different attack impact categories. The false positive rate of the pure supervised RF model is the highest in all categories except the "Crash the application" and "Escalate privilege level" categories. Furthermore, pure supervised models, and our SHIL models all achieve zero false positive rate in the "Crash the application" category. CDL-95% model has the highest false positive rate in "Crash the application" and "Escalate privilege level" categories and the second highest false positive rate in the remaining categories. The false positive rate of CDL-99% and Self-patch model is generally low, but its false positive rate in "Crash the application" category is 4.23% and 2.40% respectively. SHIL achieves consistently low false positive rate in all categories. Among all threat impact categories, the false positive rate of SHIL is below 1% in 31 attacks.

Figure 4b compares the detection rate of different schemes. SHIL achieves over 87.5% detection rate, except in "Disclose credential information" and "Crash the application" categories. In the "Execute arbitrary code" category, SHIL achieves perfect detection rate. The pure supervised CNN model can only detect 12.5% attack belonging to the "Crash the application" vulnerabilities while pure supervised RF model is only able to detect 25% of attacked containers. The reason is that the attacks in this category happen in such a short time that the pure supervised models cannot get enough training samples. However, the pure unsupervised model can detect some of the attacks in this category that show high reconstruction errors. SHIL uses the detection from the unsupervised model and does not always need to validate with the supervised model because the reconstruction errors from the detected samples are usually much higher than the boundary case. Thus, the detection rate of SHIL is still higher than those of the supervised models.

Next, we compare the lead time of different models shown in Figure 4c. Overall, the lead time is similar among all different models except pure supervised CNN model having a small lead time in the "Return a shell and execute arbitrary code" category. All six models achieve a large lead time in the "Return a shell and execute arbitrary code" category. CDL-95% model achieves the largest lead time in the "Return a shell and execute arbitrary code", "Disclose credential information", and "Escalate privilege level" categories, while the supervised random forest achieves the largest lead time in the remaining categories. It is not surprising to see no lead time in the "Crash the application" category by all the models as the attacks suddenly terminate the applications.

System run time measurements. Table IV compares the training and testing time of CDL, Self-patch, and SHIL. SHIL takes slightly more time during training and testing due to more components in the system. However, considering each sample represents the frequency vector of all system calls produced within 100 ms, SHIL is light-weight and practical for real time security attack detection in large-scale container-based environments.

TABLE IV: System run time measurements of different learning methods. Each sample represents the frequency vector of all system calls produced within 100 ms.

System Modules	Execution Time (ms)
CDL training	7.75 ± 0.40 (5000 samples)
Self-patch training	8.20 ± 0.04 (5000 samples)
SHIL training	8.42 ± 0.42 (5000 samples)
CDL detection	7.30 ± 0.10 per sample
Self-patch detection	0.0001 ± 0.00 per sample
SHIL detection	7.65 ± 0.11 per sample

C. Case Study

In this subsection, we analyze one representative attack from the top three categories to understand how SHIL identifies the attack with lower false positive rate compared with the unsupervised model.

Return a shell and execute arbitrary code. CVE-2017-12615 occurs when the attacker uploads a malicious JSP file which contains the attack code to Apache Tomcat. Tomcat prohibits users from uploading and executing files with the suffix ".jsp" to prevent malicious operations. However, the attacker can bypass the rule by uploading a JSP file with the suffix ".jsp " containing a trailing space. After that, Tomcat re-formats the file name by removing any trailing spaces and then executes it. Compared with the unsupervised model, SHIL improves the false positive rate by 88.05% without decreasing the detection rate. The 103 false positives removed by SHIL across four containers exhibit similar patterns. The false positive samples contain the `stat` call, which has a higher frequency compared with other normal run samples. The unsupervised model identifies them as anomalies. We observe that `stat` calls occur periodically and are generated by the workload as the normal period exhibits a similar periodical pattern. SHIL's self-supervised model can identify the periodical pattern and filter out those false positives.

Execute arbitrary code. CVE-2021-44228 is an Apache Log4j vulnerability that allows an attacker to execute arbitrary expressions input via the Java naming and directory interface (JNDI) service. The attacker can send a log message with special syntax to perform a JNDI lookup of a malicious lightweight directory access protocol (LDAP) server resource. The vulnerable application using Log4j will parse the message, connect to the attacker's server, and execute the payload it receives. Compared with the unsupervised model, SHIL reduces false positive rate by 87% with no reduction in detection rate. We observe the false positive samples exhibit periodical patterns, which are generated by the dynamic periodical workload and have similar patterns to the normal execution. SHIL successfully filters out those false alarms. After the attack is triggered, we observe increasing occurrences of several system calls including `clone`, `connect`, `execve`, `fcntl` and `mmap`.

Disclose credential information. CVE-2018-15473 is an Open-SSH vulnerability that allows the attacker to steal credential information because of no limit on the maximum attempts of inputting user names and passwords. The attacker finds a valid username using a brute-force method and then cracks the passwords in a similar way.

SHIL filters out 83 false positives, reducing FPR by 92.21% without hurting the detection rate. Since the sampling interval is small, we observe certain system calls (`accept`, `stat`, `close`, `fstat`, `read` and `mmap`) do not have average distributions across normal run samples, causing further false positives.

IV. Related Work

In this section, we compare our research with related work.

Container vulnerability detection. Previous work has been done in detecting vulnerabilities within containerized environments. Lin et al. [8] studied 11 privilege escalation exploits and proposed a defense mechanism to defeat privilege escalation attacks. Self-Patch [11] combined light-weight dynamic attack detection and targeted patching to achieve effective security protection for containerized applications. CDL [10] was a classified distributed learning framework for containerized applications. Lindvärn [23] et al. proposed using isolation forest for anomaly detection to achieve good detection rate and a relatively low false positive rate for 22 attacks. DIVA [2] performed vulnerability detection on Docker Hub images. Similarly, DIVDS [3] diagnosed Docker images when they are uploaded or downloaded from Docker image repositories. SHIL complements the existing work by providing a new efficient security attack detection mechanism by combining supervised and unsupervised learning methods.

Semi-supervised learning based intrusion detection. Previous work has been done in adopting a semi-supervised learning model to generate or label data for intrusion detection. Rathore et al. [24] proposed a decentralized fog-based attack detection framework which uses the semi-supervised fuzzy c-means (ESFCM) algorithm with an extreme learning machine (ELM) to detect attacks that occurred on internet of things (IoT) devices. Idhammad et al. [12] introduced a semi-supervised learning approach for DDoS detection based on network entropy estimation, co-clustering, information gain ratio, and extra trees. Zimba et al. [25] proposed a semi-supervised algorithm based on shared nearest neighbour (SNN) clustering to detect advanced persistent threat (APT) attacks. Khonde et al. [26] described an ensemble-based semi-supervised learning approach for a distributed intrusion detection system. Compared with the existing work which started from supervised learning models, SHIL uses unsupervised models as the main decision-making modules and only employs supervised models on-demand for boundary cases to filter out potential false alarms.

Supervised learning based intrusion detection. Previous work has been done in applying supervised learning methods to intrusion detection. Anthi et al. [9] described a three-layered system using supervised learning for intrusion detection for smart home IoT devices. DTB-IDS [27] presented a decision-tree-based anomaly detection method to detect APT attacks. Aksu et al. [28] applied the Fisher Score algorithm to selecting features and fed the features into support vector machine (SVM), k-nearest neighbour (k-NN) and decision tree (DT) algorithms for intrusion detection. Hosseini et al. [29] performed incremental learning with supervised models to detect distributed denial of service (DDoS) attacks. Compared with the supervised learning methods, SHIL does not required labelled training data and only uses supervised models for false alarm filtering.

Unsupervised learning based anomaly detection. Previous work has been done in applying unsupervised learning methods to intrusion detection. Unicorn [30] used K-medoids and data provenance analysis to detect Advanced Persistent Threats (APTs). Scholkopf et al. [31] proposed a one-class support vector machine (SVM) and defined a frontier as a threshold for outlier detection. The isolation forest [16] isolated anomalies from normal data by randomly selecting a feature and a value in the possible range to split data points. Khan et al. [32] proposed a hybrid intrusion detection system by combining multiple unsupervised learning methods. In comparison, SHIL leverages unsupervised learning methods to detect a set of candidate anomalies and uses supervised learning to filter out likely false alarms produced by unsupervised learning methods using boundary case thresholds.

Nevertheless, intrusion detection systems (IDS), including SHIL, have limitations. Rosenberg et al. [33] show that attackers can use a camouflage algorithm to mislead machine learning classifiers such as decision trees and random forest. If attackers gain partial information about the model training set and features, they can modify their attacks to exhibit benign patterns. We may alleviate such attacks with measures such as training updates to the SHIL anomaly detection model. Furthermore, Shu et al. [34] present a unified framework for any program anomaly detection method and prove that there is a theoretical accuracy limit. The authors note that system call based anomaly detection is limited by a lack of knowledge of program internal information such as call stack activity. We can complement SHIL with security tools that leverage such program internal context.

V. Conclusion

In this paper, we have presented SHIL, a new self-supervised hybrid learning system for more efficiently detecting security attacks in container-based computing environments. SHIL identifies anomaly detection boundary cases as most likely false alarms and combines unsupervised and supervised machine learning methods to filter out majority of the false alarms without missing most of the true attacks. For practical deployment of supervised learning models, SHIL adopts a self-supervised learning approach to labelling training data automatically using outlier detection over a window of recent measurement samples when attack alerts are first raised. Our experimental results with real world security attacks including the recent high-impacting Log4j attack show that

SHIL can significantly reduce false alarms by up to 91% while maintaining similar detection rates compared to existing pure-supervised or pure-unsupervised methods. SHIL is lightweight and does not require manual data labelling, which makes it practical for security attack detection in container-based production environments.

VI. Data Availability

The data and the implementation of SHIL are publicly available at https://github.com/NCSU-DANCE-Research-Group/SHIL.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable feedback. This work is supported by the NSA Science of Security Lablet: Impact through Research, Scientific Methods, and Community Development under the contract number H98230-17-D-0080, and a Cisco grant. Any opinions, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] "Docker image vulnerability research," Federacy, 2017. [Online]. Available: https://www.federacy.com/docker_image_vulnerabilities

[2] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on docker hub," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 269–280.

[3] S. Kwon and J.-H. Lee, "Divds: Docker image vulnerability diagnostic system," *IEEE Access*, vol. 8, pp. 42 666–42 673, 2020.

[4] "Tesla's cryptojacking attack," 2018. [Online]. Available: https://redlock.io/blog/cryptojacking-tesla

[5] "Cve-2021-44228 detail," Dec 2021. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-44228

[6] J. Wetter and N. Ringland, "Understanding the impact of apache log4j vulnerability," Dec 2021. [Online]. Available: https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html

[7] Dec 2021. [Online]. Available: https://blog.checkpoint.com/2021/12/13/the-numbers-behind-a-cyber-pandemic-detailed-dive/

[8] X. Lin, L. Lei, Y. Wang, J. Jing, K. Sun, and Q. Zhou, "A measurement study on linux container security: Attacks and countermeasures," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 418–429.

[9] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home iot devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019.

[10] Y. Lin, O. Tunde-Onadele, and X. Gu, "CDL: classified distributed learning for detecting security attacks in containerized applications," in *Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 179–188.

[11] O. Tunde-Onadele, Y. Lin, J. He, and X. Gu, "Self-patch: Beyond patch tuesday for containerized applications," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2020, pp. 21–27.

[12] M. Idhammad, K. Afdel, and M. Belouch, "Semi-supervised machine learning approach for ddos detection," *Applied Intelligence*, vol. 48, no. 10, pp. 3193–3208, 2018.

[13] "Secure devops for containers, kubernetes, and cloud — sysdig," Sysdig, 2021. [Online]. Available: https://www.sysdig.com

[14] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[15] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[16] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[17] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for unix processes," in *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 1996, pp. 120–128.

[18] M. Liu, Z. Xue, X. Xu, C. Zhong, and J. Chen, "Host-based intrusion detection system with system calls: Review and future trends," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[19] E. Carter, "Sysdig 2019 container usage report: New kubernetes and security insights," Sysdig, 2019. [Online]. Available: https://sysdig.com/blog/sysdig-2019-container-usage-report/

[20] A. Newcomb, "Sysdig 2021 container security and usage report: Shifting left is not enough," Sysdig, 2021. [Online]. Available: https://sysdig.com/blog/sysdig-2021-container-security-usage-report/

[21] R. Vinayakumar, K. Soman, and P. Poornachandran, "Applying convolutional neural network for network intrusion detection," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017, pp. 1222–1228.

[22] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[23] M. Lindvärn and Z. Lundqvist, "Refining security monitoring techniques for container-based virtualisation environments," 2021.

[24] S. Rathore and J. H. Park, "Semi-supervised learning based distributed attack detection framework for iot," *Applied Soft Computing*, vol. 72, pp. 79–89, 2018.

[25] A. Zimba, H. Chen, Z. Wang, and M. Chishimba, "Modeling and detection of the multi-stages of advanced persistent threats attacks based on semi-supervised learning and complex networks characteristics," *Future Generation Computer Systems*, vol. 106, pp. 501–517, 2020.

[26] S. Khonde and V. Ulagamuthalvi, "Ensemble-based semi-supervised learning approach for a distributed intrusion detection system," *Journal of Cyber Security Technology*, vol. 3, no. 3, pp. 163–188, 2019.

[27] D. Moon, H. Im, I. Kim, and J. H. Park, "Dtb-ids: an intrusion detection system based on decision tree using behavior analysis for preventing apt attacks," *The Journal of supercomputing*, vol. 73, no. 7, pp. 2881–2895, 2017.

[28] D. Aksu, S. Üstebay, M. A. Aydin, and T. Atmaca, "Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm," in *International Symposium on Computer and Information Sciences*. Springer, 2018, pp. 141–149.

[29] S. Hosseini and M. Azizi, "The hybrid technique for ddos detection with supervised learning algorithms," *Computer Networks*, vol. 158, pp. 35–45, 2019.

[30] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," *arXiv preprint arXiv:2001.01525*, 2020.

[31] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support vector method for novelty detection," in *Advances in neural information processing systems*, 2000, pp. 582–588.

[32] M. A. Khan, M. Karim, Y. Kim *et al.*, "A scalable and hybrid intrusion detection system based on the convolutional-lstm network," *Symmetry*, vol. 11, no. 4, p. 583, 2019.

[33] I. Rosenberg and E. Gudes, "Bypassing system calls–based intrusion detection systems," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 16, p. e4023, 2017.

[34] X. Shu, D. D. Yao, and B. G. Ryder, "A formal framework for program anomaly detection," in *International Symposium on Recent Advances in Intrusion Detection*. Springer, 2015, pp. 270–292.