Self-Supervised Machine Learning Framework for Online Container Security Attack Detection

OLUFOGOREHAN TUNDE-ONADELE, North Carolina State University, USA
YUHANG LIN, Meta Platforms, USA *
XIAOHUI GU, North Carolina State University, USA
JINGZHU HE, ShanghaiTech University, China *
HUGO LATAPIE, Cisco, USA

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 labeled training data to achieve good attack detection accuracy. On the other hand, unsupervised machine learning methods are more practical by avoiding training data labeling requirements, but they often suffer from high false alarm rates. In this paper, we present a generic self-supervised hybrid learning (SHIL) framework for achieving efficient online security attack detection in containerized systems. SHIL can effectively combine both unsupervised and supervised learning algorithms but does not require any manual data labeling. We have implemented a prototype of SHIL and conducted experiments over 46 real world security attacks in 29 commonly used server applications. Our experimental results show that SHIL can reduce false alarms by 33-93% compared to existing supervised, unsupervised, or semi-supervised machine learning schemes while achieving a higher or similar detection rate.

CCS Concepts: • **Security and privacy → Distributed systems security**; **Intrusion detection systems**; • **Computing methodologies → Machine learning approaches**; **Distributed computing methodologies**.

ACM Reference Format:

Olufogorehan Tunde-Onadele, Yuhang Lin, Xiaohui Gu, Jingzhu He, and Hugo Latapie. 2024. **Self-Supervised Machine Learning Framework for Online Container Security Attack Detection.** ***ACM Trans. Autonom. Adapt. Syst.*** **1, 1, Article 1 (January 2024), 28 pages. https://doi.org/10.1145/3665795**

1 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, 23, 38] have shown that containers are vulnerable to various security attacks. For example, Tesla suffered a cryptojacking attack in February 2018 [2]. 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

*Author contributed while a PhD student at North Carolina State University.

Authors' addresses: Olufogorehan Tunde-Onadele, North Carolina State University, USA, oatundeo@ncsu.edu; Yuhang Lin, Meta Platforms, USA *, yuhangl@meta.com; Xiaohui Gu, North Carolina State University, USA, xgu@ncsu.edu; Jingzhu He, ShanghaiTech University, China *, hejzh1@shanghaitech.edu.cn; Hugo Latapie, Cisco, USA, hlatapie@cisco.com.

Manuscript submitted to ACM

affected distributed systems worldwide, including container clusters [3]. 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 [43]. In just four days after the disclosure, Check Point Research reported over 800,000 global attacks to the vulnerability [30].

Traditional intrusion detection systems [25] 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 [8, 18, 26, 41] 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 labeled 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 labeled training data. However, due to the lack of labeled training data, anomaly detection methods often suffer from high false alarms. Previous work also proposed semi-supervised learning methods [18] 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 unlabeled data. However, the semi-supervised approach still requires sufficient labeled training data in order to train the initial supervised model.

In this paper, we present a new generic *self-supervised hybrid learning* (SHIL) framework for performing online security attack detection in containerized applications. SHIL aims at improving security detection accuracy without requiring any labeled 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. To the best of our knowledge, SHIL makes the first step in combining unsupervised learning with supervised learning for cross-validations in online security attack detection without requiring any labeled training data.

SHIL leverages non-intrusive, light-weight system call monitoring tools [4] to achieve practical security attack detection for containers. SHIL consists of three key modules: 1) *unsupervised anomaly detection*, which leverages an unsupervised learning method such as autoencoder neural networks [7] to achieve fast attack detection without requiring any labeled training data; 2) *hybrid alert validation*, which identifies boundary case anomalies and performs false alarm filtering by cross validating the anomalies using a supervised learning method such as Random Forest [16]; and 3) *self-supervised model creation*, which performs automatic data labeling using outlier detection (e.g., isolation forest [28]) and similarity filtering over a window of recent system call data upon an attack alert raised by either SHIL or a third party attack detection tool.

Specifically, this paper makes the following contributions:

- We propose a new self-supervised hybrid machine learning framework which combines unsupervised and supervised learning methods to achieve effective security attack detection with few false alarms.
- We describe a self-supervised model creation method which leverages both outlier detection and similarity filtering to produce training labels for supervised learning methods automatically.

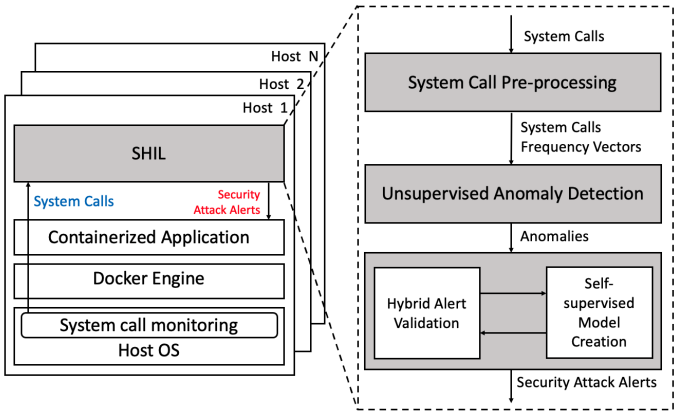


Fig. 1. System overview of SHIL.

- We implement and evaluate a prototype of SHIL over 46 recent real critical vulnerabilities with high CVSS scores including the high impact Apache Log4j vulnerability, in 29 commonly used server applications.
- We implement a set of pure-supervised, pure-unsupervised, and semi-supervised machine learning based security attack detection schemes for comparative study. Our experimental results show that SHIL can reduce false alarms by 33% to 93% compared to existing supervised, unsupervised, or semi-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 2 presents the system design in detail. Section 3 describes our experimental evaluation methodology and our experimental results. Section 4 compares our work with related work. Section 5 concludes this paper.

2 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 [4] 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 [12, 29]. 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). Thus, our definitions of normality is that the system call frequency vector values stay in the common range, that is, being similar to other system call frequency vectors during normal execution time. Since we observe that most attacks we studied manifest as significant frequency changes over certain system calls during the attack period, we use abnormal system call frequency vector values to detect an attack occurrence. Interestingly, not all system call frequency

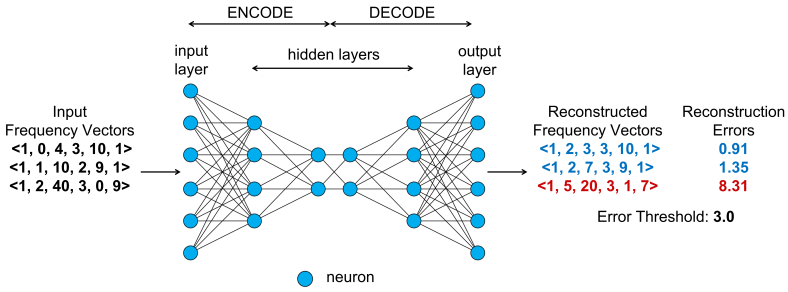


Fig. 2. An example of anomaly detection with an autoencoder. A frequency vector with a reconstruction error that exceeds the threshold is considered an anomaly.

vector values exhibit abnormality during the attack period because the attack can adopt a non-persistent strategy. Hence, it is important to filter out those normal system call frequency vectors during the attack period.

Table 1 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 t_0 + 400 and ends at timestamp t_0 + 1800. 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 [12]) 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 2.1 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 2.2 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 [28]. Section 2.3 will provide details about this module.

2.1 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

Table 1. A frequency vector sample for the *OpenSSH* application (CVE-2016-6515). An attack is triggered at t = t_0 + 400. Anomaly detection raises alarms from t = t_0 + 600 to t = t_0 + 700. The entries with asterisks (*) are detected outliers.

Timestamp (milliseconds)	System Call Frequency			
	access	**execve**	**lstat**	**mmap**
t_0 = 1586496672491	0	0	0	0
t_0 + 100	0	0	0	0
t_0 + 200	0	0	0	0
t_0 + 300	0	0	0	0
t_0 + *400* (attack starts)	*0*	*0*	*0*	*0*
t_0 + 500	79	7	0	155
t_0 **+ 600** (attack detected)	**136**	**41**	**12**	**420**
t_0 **+ 700***	**268**	**51**	**16**	**702**
t_0 + 800*	209	46	24	637
t_0 + 900	164	23	8	422
t_0 + 1000	70	32	24	290
t_0 + 1100	190	21	12	454
t_0 + 1200	76	12	12	297
t_0 + 1300*	129	55	28	386
t_0 + 1400	130	5	0	353
t_0 + 1500	82	20	4	249
t_0 + 1600*	159	42	20	439
t_0 + 1700	79	3	8	260
t_0 + 1800 (attack succeeds)	91	53	16	250
t_0 + 1900	192	12	0	468
t_0 + 2000	50	1	0	142
t_0 + 2100	0	0	0	0
t_0 + 2200	0	0	0	0

the unsupervised learning methods by only involving supervised models for performing cross validations on boundary cases.

Our unsupervised anomaly detection leverages autoencoder neural networks, shown in Figure 2. 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. In our previous work, we extensively compared autoencoder-based anomaly detection algorithms with other alternative anomaly detection algorithms and dimensionality reduction methods such as principal component analysis (PCA) or self-organizing map (SOM) [40, 41]. We found auto-encoder based anomaly detection can achieve higher anomaly detection accuracy than other alternatives.

During the 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 [26].

2.2 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 to detect 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.

2.3 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 (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 1, 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 3. 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 pre-process 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

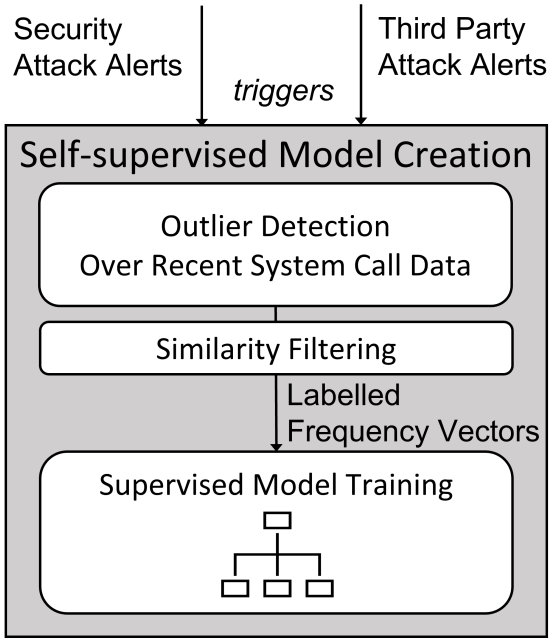


Fig. 3. Self-supervised model creation.

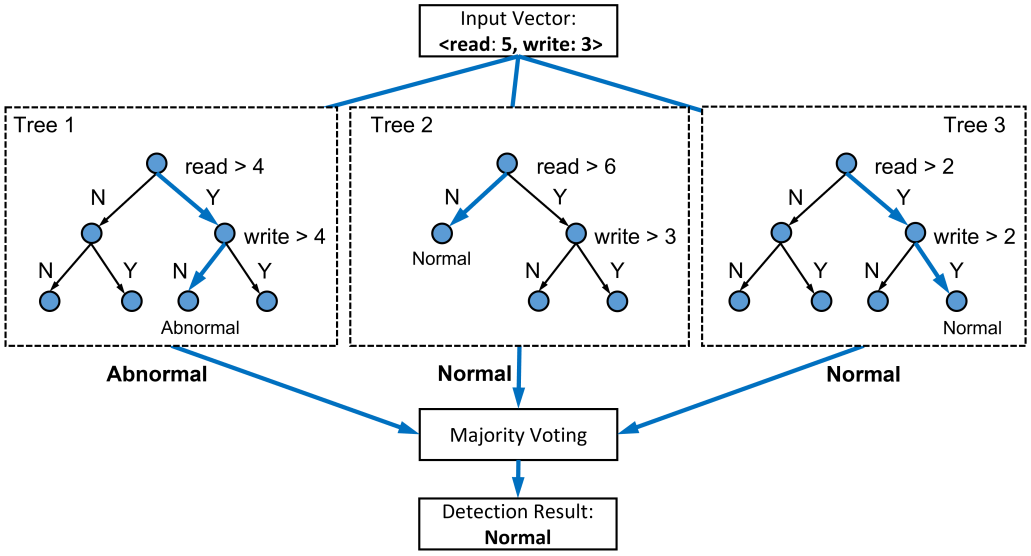


Fig. 4. An example of detection with the random forest supervised model. The final decision is based on the majority vote from all the trees.

attack period with all the normal measurement samples collected during a small window preceding the attack detection time. We then further filter those outliers which resemble the normal execution behavior. The rationale behind our

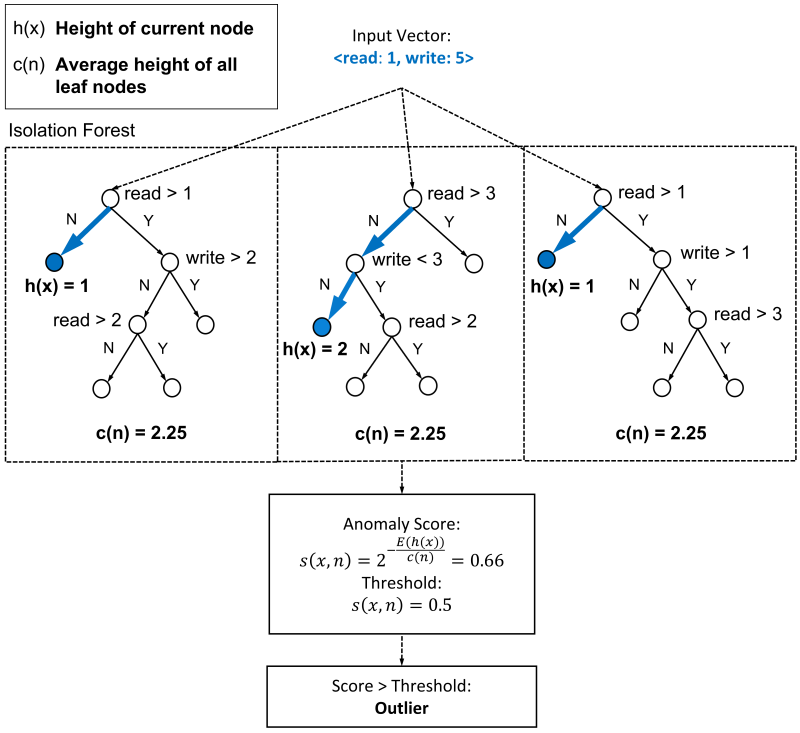


Fig. 5. An example of outlier detection with the isolation forest model. The leaf nodes with the shortest heights from their root nodes tend to be outliers.

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 describe each step in detail in the following subsections.

2.3.1 Supervised Model. SHIL is a generic self-learning hybrid learning framework which can be applied to any supervised and unsupervised learning methods. For extensive comparative study, we select three commonly used supervised learning methods: Random forest(RF), Convolutional neural network(CNN), and Recurrent neural network(RNN), which are described as follows.

Random forest(RF): The RF model, depicted in Figure 4, 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 noise. For a single tree, the fraction of abnormal sample predictions in the output leaf node yields a probability value. The overall prediction probability of a node is the average prediction probability over all its

decision trees. When the prediction probability is above a pre-defined threshold such as 60%, the frequency vector is classified as abnormal.

Convolutional neural network(CNN): The convolutional neural network architecture consists of convolution layers that traverse over different regions of input features to learn relevant patterns. A matrix of weights, referred to as a kernel, is given a certain size to filter the region that the convolution is performed over. The output of the convolution as the kernel moves over the input data feeds to the next layer, which helps the network learn hierarchical patterns as the process is repeated over multiple layers. Thus, the CNN performs valuable feature selection and has been applied especially in domains such as image classification. However, using the CNN incurs the cost of examining many feature combinations to learn useful relationships. Furthermore, the CNN does not effectively capture temporal information. We construct a CNN composed of one-dimensional (1D) convolutional layers with rectified linear unit (ReLU) activation, linked to a flatten layer before the final interconnected dense layer with sigmoid activation. We readily apply the input frequency vectors to the neural network to output the probability of the vector being abnormal. We train the network weights with the Adam optimization algorithm [22]. The specific implementation details are discussed in the experimental evaluation.

Recurrent neural network(RNN): The recurrent neural network is a neural network architecture where a node output can feed back to the node to allow memory of data from previous time steps. This feedback capability allows the RNN to learn sequential patterns from time-series data. However, the RNN may forget information from previous steps due to the vanishing gradient problem. Therefore, it often demands extensive training to attain strong results. We implement an RNN architecture consisting of stacked recurrent layers with hyperbolic tangent (tanh) activation, joined to a final dense layer with sigmoid activation. The recurrent layers accept an input frequency vector sequence with previous time-steps and then the dense layer outputs the probability of the input sequence being abnormal. As with the CNN, we train the network weights according to the Adam algorithm.

2.3.2 Outlier Detection. We employ the isolation forest [28] 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 5 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.

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

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 expected value 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 [28]. 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 5, 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 1 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. However, we observe that using the outlier detection scheme only did not always improve the hybrid alert validation results. Since the data from the outlier detection are used to train the supervised model to distinguish normal and abnormal samples, the supervised model depends on the outlier detection for true labels to properly validate the boundary cases presented by the anomaly detection component. We observe that certain system call data labelled as attack samples resemble that of normal data, which confuses the supervised model with conflicting labels. Therefore, we further improve the labels by filtering out the misleading attack samples that resemble normal data so they are not used to train the supervised model.

2.3.3 Similarity Filtering. We perform similarity filtering to extract true anomalies from the initial outlier detection algorithm. Specifically, we identify those rogue outliers during the attack period by comparing all outliers with the normal data outside the attack period. To measure the similarity between two frequency vectors, we employ the Manhattan distance, given by the following equation. For two frequency vectors, $a = [a_1, ...a_i, ...a_N]$ and $b = [b_1, ...b_i, ...b_N]$, the Manhattan distance d is:

$$d = \sum_{i=1}^{N} |a_i - b_i| \tag{2}$$

We choose the Manhattan distance as it is preferable for high-dimensional data by providing better relative contrast among distances to a query data point compared to the more common Euclidean distance [5]. Those outliers that are similar to normal data outside the attack period are labelled as normal data to train the supervised model.

SHIL provides a generic self-supervised hybrid learning framework which can be applied to different machine learning schemes. Users can choose different machine learning algorithms based on their security attack detection goals such as maximizing detection rate or minimizing false alarm rate. During our experiments, we observe that the random forest learning method can achieve the best trade-off across detection rate, false positive rate, and training time. In contrast, the neural network approaches such as convolutional neural networks (CNN) [42] and recurrent neural networks (RNN) [36] require a substantial amount of high quality training data and incur higher training time. We conduct extensive comparative study in section 3 to show the effectiveness of different machine learning schemes.

3 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.

3.1 Evaluation Methodology

Real-world vulnerabilities. We evaluate 46 vulnerabilities from 29 applications listed in the common vulnerabilities and exposures (CVE) database, which include many applications commonly used in production environments [9, 32]. Table 2 shows the complete list of the CVEs, including the common vulnerability scoring system (CVSS) score v2.0,

Table 2. 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-2017-7504	7.5	JBoss	4.05
	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-2019-0193	9.0	Apache Solr	8.1.1
	CVE-2019-0230	7.5	Apache Struts 2	2.5.16
	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	9.3	Apache Solr (Log4j)	8.11.0
	CVE-2022-24706	10.0	CouchDB	3.2.1
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
	CVE-2018-10933	6.4	Libssh	0.8.1

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.

Experiment Setup. For each vulnerability, we set up four Docker containers under Ubuntu 18.04 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 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. The sampling rate granularity is chosen to achieve a good trade-off between the pre-processing overhead and sufficient data samples for real-time detection [40]. For instance, finer granularity produces smaller-sized samples but makes the detection system more heavyweight. We primarily split the data so that the first three minutes are for training and the last four minutes are for testing. Therefore, there is no overlap between the training and testing data sets. The attack lasts from the fifth minute until it completes so the testing data contains both normal and attack samples.

SHIL Prototype Implementation (SHIL-RF). Our autoencoder (AE) model configuration arises from the tuning experiments performed in our prior work [26]. Specifically, we use TensorFlow to build the autoencoder 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. The autoencoder model measures the reconstruction error using root mean square error (RMSE) as defined in Equation (3), where N is the number of samples, y_i is the value of input, and $\hat{y}_i$ is the value of the output. The RMSE loss function between frequency vector values is simple to compute and interpret.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}{N}} \tag{3}$$

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 as described in subsection 2.3. 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 a similarity distance threshold of five.

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 it with several alternative real-time, light-weight security attack detection methods CDL and Self-Patch [26, 41] that have been proposed for container systems. We also implement pure-unsupervised, pure-supervised, and semi-supervised methods in addition to hybrid learning variations for comparative study.

- **Classified Distributed Learning (CDL) [26]**: We run CDL using 95 and 99 percentile anomaly detection thresholds. CDL uses the same autoencoder model as used in SHIL.

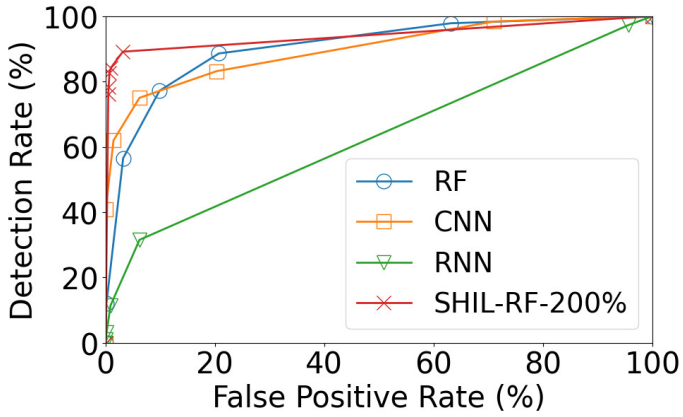


Fig. 6. Sensitivity of the supervised models based on the best results from our tuning experiments.

- **Self-Patch [41]**: 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 a 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. We illustrate the sensitivity of all the supervised models at various confidence levels with the receiver operating characteristic (ROC) curve in Figure 6 using the best results from our tuning experiments.
- **Supervised (CNN)**: We use the convolution neural network (CNN) learning method, which has been recently applied to cybersecurity [42]. We use Keras, with Tensorflow as the backend, to implement this model. We obtain the following model after tuning the parameters, including the number of layers, neurons per layer, and confidence. This model consists of two 1-D convolution layers with the rectified linear activation function (ReLU) and a kernel size of 10. 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 and second CNN layers have 64 and 32 filters, respectively. 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.
- **Supervised (RNN)**: Similarly, we test the recurrent neural network (RNN) to examine whether the model can leverage temporal insights in the dataset [36]. We implement the RNN with Keras and Tensorflow. After experimenting with the architecture and hyperparameters, the final RNN model comprises four stacked RNN layers of 512 recurrent units each and a final dense layer. The recurrent and dense layer groups use the hyperbolic tangent (tanh) and sigmoid activation functions, respectively. We ultimately train the neural network with a sequence length of 10 time steps and 30 training iterations using binary cross-entropy loss.

- **Semi-supervised**: The semi-supervised method starts with the supervised RF model and then uses the unsupervised autoencoder, compared to SHIL that starts with the unsupervised autoencoder model and uses a supervised RF model to improve detection. We employ the self-training classifier [44] using the implementation available in scikit-learn. We conduct our study with the same dataset used in the supervised model. However, we randomly masked 30% of the data as unlabeled, to be used as the input for the self-training classifier. The classifier functions by iteratively making predictions on unlabeled data, which are then added to the training set for retraining. To ensure fair comparison, the random forest algorithm was chosen as the base estimator and the hyperparameters used were identical to those of the random forest model employed in SHIL.
- **Similarity filtering (SF) Only**: We examine the similarity filtering approach alone for attack detection. If the sample is similar to a sample in the training data, the sample is considered as normal; otherwise, the sample is labeled as abnormal. We use the same similarity distance measure as in the SHIL prototype discussed above but adjust the threshold to 75 for comparable detection results.
- **Hybrid-SF Only**: We also examine the hybrid learning architecture with the similarity filtering only algorithm as the self-supervised model. Thus, in the boundary case, the cross validation result between that of the unsupervised model and that of the similarity filtering only algorithm is the final decision for the sample. The similarity distance settings are the same as that of the similarity filtering only method.
- **Hybrid-CNN**: We modify the self-supervised hybrid learning architecture with two changes. First, we select the CNN model described above instead of the random forest as the supervised model. Second, we do not perform outlier detection or similarity filtering during the supervised model creation so the training data samples during the attack alert window are all labeled as abnormal.
- **Hybrid-CNN-OD**: This alternative is the Hybrid-CNN with outlier detection only. Here, the outlier detection labels the outliers during the attack alert period as positive attack samples and non-outliers as normal samples.
- **SHIL-CNN**: This approach uses the CNN as the supervised model in the SHIL framework, which is the same as the Hybrid-CNN with outlier detection and similarity filtering. During the attack alert window, we label the samples that are outliers and are not similar to normal data samples as the positive attack samples. All other samples in this period are labeled as normal.
- **Hybrid-RNN**: Similar to the Hybrid-CNN, we construct a hybrid learning alternative in which we replace the random forest supervised model with the supervised RNN and do not perform outlier detection or similarity filtering.
- **Hybrid-RNN-OD**: This alternative is the Hybrid-RNN with outlier detection only. Here, the outlier detection labels the outliers during the attack alert period as positive attack samples and non-outliers as normal samples.
- **SHIL-RNN**: This approach uses the RNN as the supervised model in the SHIL framework, which is the same as the Hybrid-RNN but with both outlier detection and similarity filtering. During the attack alert window, we label the samples that are outliers and are not similar to normal data samples as the positive attack samples. All the other samples in this period are labeled as normal.
- **Hybrid-RF**: This hybrid learning approach uses the random forest supervised model but does not perform outlier detection or similarity filtering.
- **Hybrid-RF-OD**: This alternative is the Hybrid-RF with outlier detection only so the outlier detection component labels the outliers during the attack alert period as positive attack samples and the others as normal samples.
- **SHIL-RF**: This is the default SHIL prototype, which uses the random forest supervised model with outlier detection and similarity filtering.

Table 3. Comparison with alternative approaches.

Model	Detection rate	FPR	Lead time
CDL-95%	90.76% ± 0.77 (p = 0.00047e−4)	6.86% ± 0.39 (p = 0.2600e−8)	9.31s ± 0.04
CDL-99%	81.08% ± 0.97 (p = 0.00360e−4)	1.27% ± 0.17 (p = 0.0053e−8)	8.44s ± 0.34
Self-Patch	46.20% ± 0.00 (p = 0.00000e−4)	1.38% ± 0.00 (p = 0.0000e−8)	7.14s ± 0.00
Supervised (RF)	77.17% ± 0.00 (p = 0.00000e−4)	9.88% ± 0.00 (p = 0.0000e−8)	9.28s ± 0.00
Supervised (CNN)	73.59% ± 1.66 (p = 0.09100e−4)	5.36% ± 0.19 (p = 0.0140e−8)	7.50s ± 0.24
Supervised (RNN)	33.59% ± 3.49 (p = 7.12000e−4)	6.45% ± 0.92 (p = 7.4700e−8)	7.59s ± 0.32
Semi-supervised	63.59% ± 0.00 (p = 0.00000e−4)	1.07% ± 0.00 (p = 0.0000e−8)	7.42s ± 0.00
Similarity Filtering Only	90.22% ± 0.00 (p = 0.00000e−4)	24.87% ± 0.00 (p = 0.0000e−8)	9.39s ± 0.00
SHIL-RF 120% Boundary	**88.59% ± 0.00** (p = 0.00000e−4)	**1.98% ± 0.00** (p = 0.0000e−8)	**8.65s ± 0.00**
SHIL-RF 200% Boundary	**82.61% ± 0.00** (p = 0.00000e−4)	**0.72% ± 0.00** (p = 0.0000e−8)	**7.73s ± 0.00**

We configuered CDL and Self-Patch models based on the previous work as they have been tuned before by the authors. For all the other models (e.g., CNN, RF, RNN), we experiment with different parameter values to show best trade-off between detection rate and false positive rate.

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 (4) and (5), respectively.

$$detection\ rate = \frac{DC}{DC + MC} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

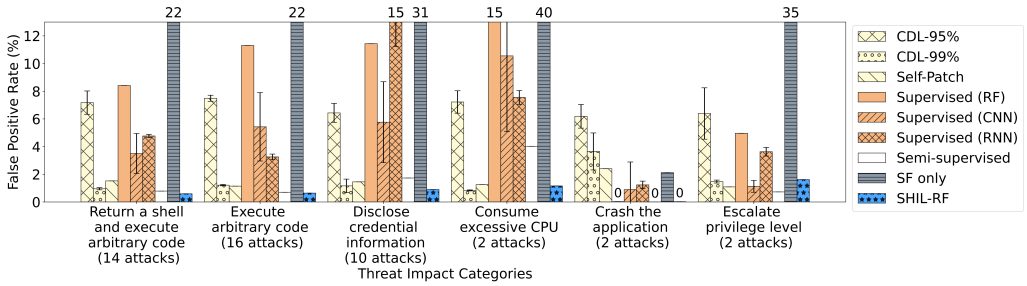
We choose not to use sample count based true positive metrics since not all samples during the attack period are abnormal. Moreover, our approach aims at raising alerts as early as possible when we detect the first anomalous attack behaviors during the attack period. We introduce *lead time* to denote 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. We observe an attack as successful when its command completes at the terminal.

3.2 Results Analysis

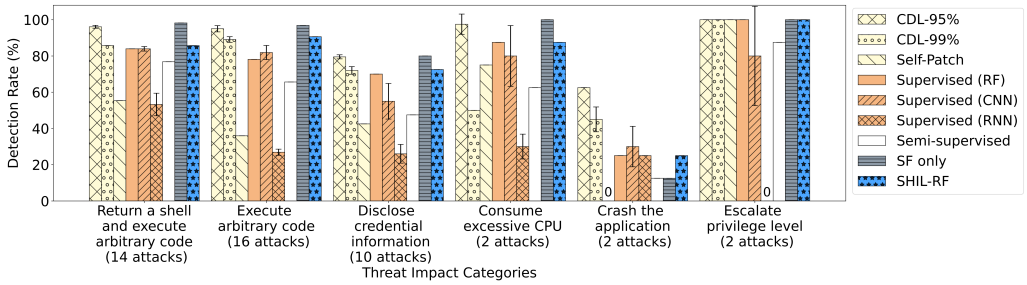
In this section, we present the analysis of the experiment results. We repeat each experiment five times and report both mean and standard deviation results. In addition, we perform the t-test statistical test. Here, the null hypothesis is that the models detect the attacks no better than random chance (i.e. mean detection rate and FPR of 50%). For each model, we calculate the t-score according to Equation 6 below, where sm is the mean metric value over the experiment trials, m is the random chance mean, σ is the standard deviation value over the experiment trials, and n is the number of experiment trials.

$$t\text{-}score = \frac{sm - m}{\frac{\sigma}{\sqrt{n-1}}} \tag{6}$$

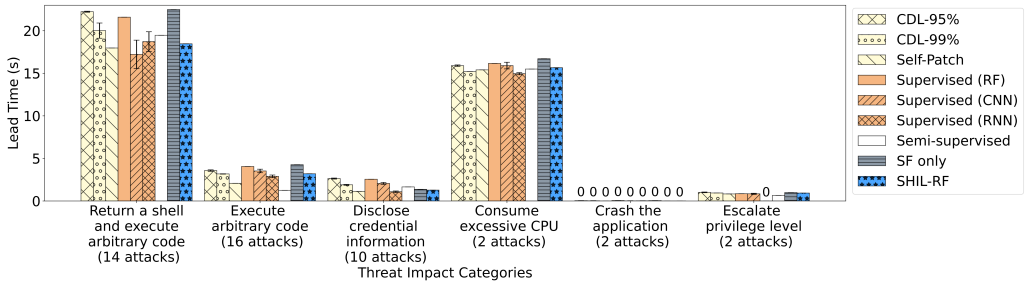
Thereafter, we extract the corresponding p-value of the two-tailed t-test. We present the p-value results in the table and select a strong significance value (alpha) of 0.01 (99% confidence). All p-values are below 0.01 so we reject the null



(a) False positive rate comparison.



(b) Detection rate comparison.



(c) Lead time comparison.

Fig. 7. Model comparisons among unsupervised autoencoder using 95% and 99% thresholds, supervised RF, supervised CNN, supervised RNN, Semi-supervised, and SHIL-RF (200% Boundary).

hypothesis that the models attack detection the attacks no better than random chance. Moreover, all p-values in all our experiments are lower than 0.00072.

Table 3 compares the detection results among CDL [26] using 95 percentile (CDL-95%) and 99 percentile (CDL-99%) anomaly detection thresholds, Self-Patch [41], the pure supervised models (RF, CNN, RNN), the semi-supervised models and our SHIL-RF approach using 120% (SHIL-RF-120%) and 200% boundary case threshold (SHIL-RF-200%).

For CDL, increasing the anomaly detection threshold from 95 percentile to 99 percentile reduces the FPR from 6.86% to 1.27%. However, we also see a substantial drop in detection rate from from 90.76% to 81.08%. Self-Patch has a similar FPR to CDL-99%, but its detection rate yields only 46.20%. In contrast, SHIL-RF-120% model can reduce FPR significantly by 71.1% while maintaining similar detection rate and lead time compared to the pure unsupervised model CDL-95%. By increasing the boundary cases to a higher threshold, SHIL-RF-200% can achieve even better detection rate while

Table 4. Comparison of different self-supervised hybrid approaches using 120% Boundary.

Approach	Outlier Detection	Similarity Filtering	Detection Rate	FPR	Lead Time
Hybrid SF Only	✗	✓	88.04% ± 0.00 (p = 0.000e−7)	3.25% ± 0.00 (p = 0.000e−9)	8.78s ± 0.00
Hybrid-CNN	✗	✗	86.41% ± 0.00 (p = 0.580e−7)	2.36% ± 0.20 (p = 0.120e−9)	9.03s ± 0.07
Hybrid-CNN-OD	✓	✗	86.52% ± 0.46 (p = 0.092e−7)	2.08% ± 0.01 (p = 0.000e−9)	8.99s ± 0.04
SHIL-CNN	✓	✓	85.87% ± 0.00 (p = 0.000e−7)	1.94% ± 0.01 (p = 0.000e−9)	8.92s ± 0.01
Hybrid-RNN	✗	✗	86.96% ± 0.55 (p = 9.510e−7)	2.03% ± 0.12 (p = 0.013e−9)	8.84s ± 0.17
Hybrid-RNN-OD	✓	✗	86.19% ± 0.30 (p = 0.054e−7)	1.94% ± 0.03 (p = 0.000e−9)	8.75s ± 0.27
SHIL-RNN	✓	✓	85.87% ± 0.54 (p = 0.190e−7)	1.94% ± 0.00 (p = 0.000e−9)	8.52s ± 0.33
Hybrid-RF	✗	✗	89.13% ± 0.00 (p = 0.000e−7)	2.53% ± 0.00 (p = 0.000e−9)	8.79s ± 0.00
Hybrid-RF-OD	✓	✗	88.59% ± 0.00 (p = 0.000e−7)	2.14% ± 0.00 (p = 0.000e−9)	8.65s ± 0.00
SHIL-RF	✓	✓	**88.59% ± 0.00** (p = 0.000e−7)	**1.98% ± 0.00** (p = 0.000e−9)	**8.65s ± 0.00**

Table 5. Comparison of different self-supervised hybrid approaches using 200% Boundary.

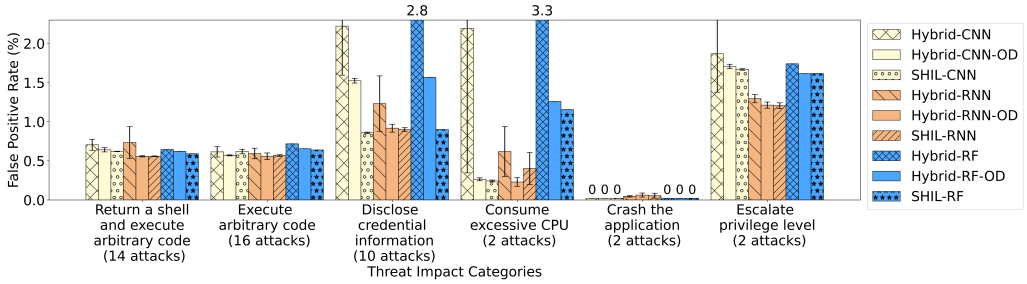
Approach	Outlier Detection	Similarity Filtering	Detection Rate	FPR	Lead Time
Hybrid SF Only	✗	✓	83.70% ± 0.00 (p = 0.000e−7)	2.73% ± 0.00 (p = 0.000e−9)	8.61s ± 0.00
Hybrid-CNN	✗	✗	79.78% ± 0.59 (p = 0.058e−7)	1.09% ± 0.20 (p = 0.100e−9)	7.57s ± 0.06
Hybrid-CNN-OD	✓	✗	81.41% ± 0.71 (p = 0.990e−7)	0.81% ± 0.00 (p = 0.000e−9)	7.88s ± 0.29
SHIL-CNN	✓	✓	80.65% ± 0.30 (p = 0.035e−7)	0.67% ± 0.01 (p = 0.000e−9)	7.76s ± 0.38
Hybrid-RNN	✗	✗	78.80% ± 1.15 (p = 9.510e−7)	0.78% ± 0.17 (p = 0.050e−9)	7.98s ± 0.23
Hybrid-RNN-OD	✓	✗	77.50% ± 0.30 (p = 0.054e−7)	0.63% ± 0.02 (p = 0.000e−9)	7.58s ± 0.62
SHIL-RNN	✓	✓	77.07% ± 0.71 (p = 1.760e−7)	0.63% ± 0.01 (p = 0.000e−9)	7.27s ± 0.63
Hybrid-RF	✗	✗	82.61% ± 0.00 (p = 0.000e−7)	1.27% ± 0.00 (p = 0.000e−9)	7.93s ± 0.00
Hybrid-RF-OD	✓	✗	82.07% ± 0.00 (p = 0.000e−7)	0.88% ± 0.00 (p = 0.000e−9)	7.73s ± 0.00
SHIL-RF	✓	✓	**82.61% ± 0.00** (p = 0.000e−7)	**0.72% ± 0.00** (p = 0.000e−9)	**7.73s ± 0.00**

reducing the FPR by 43.3%, 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 to achieve lower FPR at similar or better detection rate.

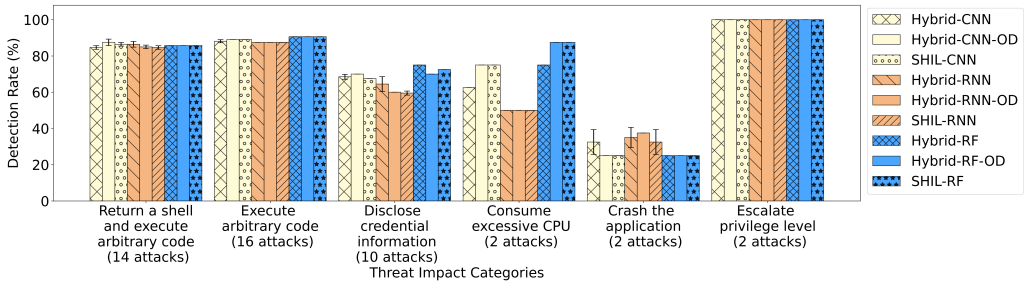
The pure supervised models (RF, CNN, RNN) suffer from both low detection rate and high false alarms due to low quality labeled training data [19]. When comparing with the pure-supervised RF and CNN models in the table, SHIL-RF-200% has 92.7% and 86.6% lower FPR, respectively with higher detection rate. The pure-supervised RNN model especially suffers from low detection rate. Although its FPR is lower than the pure-supervised RF model, its detection rate is notably low due to limited training data. We represent the sensitivity of the supervised models with the ROC curve given in Figure 6 using the best results over different confidence levels. Our investigations show that adjusting the RNN confidence offers tuning flexibility, albeit at lower confidence levels. To build a more effective model, the recurrent network needs a large amount of training data, which may not be not readily available in container-based distributed environments. Nevertheless, SHIL-RF-200% reduces the FPR by 88.8%. Compared to the semi-supervised learning method, SHIL-RF-200% reduces the FPR by 32.7% with a much higher detection rate and a longer lead time. This result shows that traditional hybrid learning scheme suffers from both high false positive and low detection rate due to the lack of sufficient high quality labeled training data.

Moreover, similarity filtering only does not perform well as it has high FPR of 24.87%. Using SHIL-RF 120%, however, gives 92.0% lower FPR with similar detection rate. Thus, we find that it is not adequate to identify normal and attack samples simply based on similarity to training data samples.

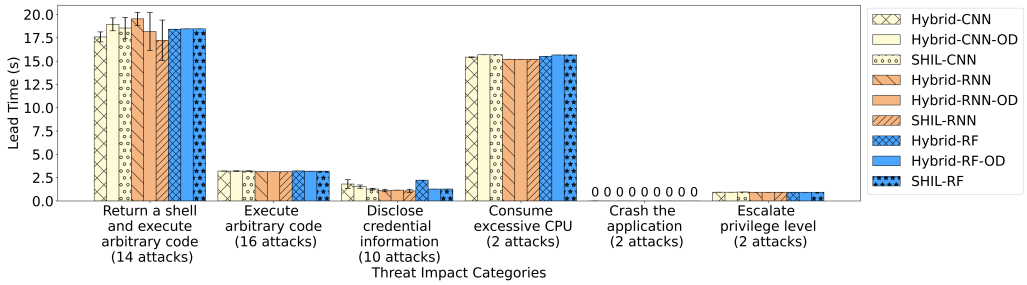
Figure 7a shows the FPR of different models under different attack impact categories. As one might expect from the overall results, the false positive rate of the similarity filtering is the highest across almost all categories, except "Crash



(a) False positive rate comparison.



(b) Detection rate comparison.



(c) Lead time comparison.

Fig. 8. Comparisons among the different self-supervised hybrid approaches.

the application". Thereafter, the pure-supervised RF model is typically the next highest. Among the pure-supervised models in these categories, the RNN usually yields the next highest FPR, followed by the CNN. Furthermore, most pure-supervised and hybrid models show nearly zero FPR for the "Crash the application" category. The CDL-95% model has the highest FPR in this category; its value tends to stay at similar levels across the threat impact groups. The false positive rates of CDL-99% and the Self-Patch model are generally low, but their values in the "Crash the application" category are 3.65% and 2.40% respectively. The semi-supervised model exhibits low FPR results below 2%, except the 4.02% value in the "Consume excessive CPU" category. SHIL-RF achieves consistently low FPR in all categories. Among all threat impact categories, the FPR of SHIL-RF stays below 1.7%.

Next, Figure 7b compares the detection rate of the different schemes. SHIL-RF attains relatively high detection rate. It generally achieves over 85% detection rate, except in the "Disclose credential information" and "Crash the application" categories. In the "Execute arbitrary code" and "Escalate privilege level" categories, SHIL-RF achieves over 90% detection

rate and 100% detection rate, respectively. Most models are only able to detect up to 25% of attacks belonging to the "Crash the application" vulnerabilities. The reason is that the attacks in this category happen in such a short time that the models cannot get enough training samples. However, the pure unsupervised models can detect some of the attacks in this category that show high reconstruction errors but at the cost of high FPR. Overall, the detection rate of SHIL-RF is higher than that of the supervised models across the threat impact categories. 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.

Finally, we compare the lead time of the different models shown in Figure 7c. Overall, the lead time is similar among the different models in the same threat impact category. However, the lead time of the pure-supervised RNN model is the lowest in three categories, namely "Disclose credential information", "Consume excessive CPU", and "Escalate privilege level". All nine models achieve a large lead time in the "Return a shell and execute arbitrary code" category as the attacks take more time to successfully return the reverse shell. The CDL-95% and the supervised RF models achieve the largest lead time in the "Disclose credential information" and "Crash the application" categories, while SF-only achieves the largest lead time in the remaining categories. It is not surprising to see minimal lead time in the "Crash the application" category by all the models as the attacks suddenly terminate the applications.

Self-supervised model learning comparative study. Furthermore, we conduct a sensitivity study about different learning steps in SHIL. Table 4 and Table 5 summarize the overall results of the hybrid methods using the CNN, RNN, and RF as the supervised models. Table 4 outlines the results of the 120% boundary and Table 5 gives results of the 200% boundary. In these tables, we also give the results of the Hybrid SF only approach to show baseline hybrid results with the naive similarity filtering only method.

Recall that without outlier detection, all samples within the attack alert period are labeled as abnormal for supervised model training. When the outlier detection component directly labels the samples without any filtering, the FPR results generally improve with comparable detection rate and lead time. Specifically, for the 120% boundary, the outlier detection results in up to 15.4% lower FPR for the Hybrid-CNN, Hybrid-RNN, and Hybrid-RF. Instead, the detection rates of the Hybrid-CNN, Hybrid-RNN, and Hybrid-RF only slightly change by 0.13%, 0.89%, and 0.61%. The change yields similar or up to 1.6% shorter lead time. For the 200% boundary, the Hybrid-CNN-OD and Hybrid-RF-OD result in 25.7% and 30.7% lower FPR respectively, while Hybrid-RNN-OD gives a smaller drop of 19.2% FPR. The change also results in similar or up to 5.0% shorter lead time, except for the Hybrid CNN whose lead time rises by about 4.1%. The detection rates of the Hybrid-RNN and Hybrid-RF drop by 1.65% and 0.65% respectively but increase for the Hybrid-CNN by 2.0%.

When similarity filtering is further executed, the FPR significantly improves, especially compared to that of the hybrid methods without outlier detection. Compared to the Hybrid-RF using 120% boundary, the SHIL-RF FPR improves by 21.7% with similar detection rate and lead time. Next, the SHIL-CNN attains 17.8% better FPR results, with 0.6% and 1.2% lower detection rate and lead time, respectively. Finally, the SHIL-RNN yields 4.4% lower FPR and 3.6% shorter lead time with a 1.3% reduction in detection rate. Furthermore, SHIL reduces FPR more considerably with the larger 200% boundary. The SHIL-RF FPR improves by 43.3% with no reduction in detection rate and similar lead time. Next, the SHIL-RNN yields 19.2% lower FPR with a 2.2% and 8.9% reduction in detection rate and lead time, respectively. Finally, the SHIL-CNN attains 38.5% better FPR results, 1.1% better detection rate, and 2.5% better lead time. Overall, with the SHIL framework, the user can select the supervised model that best aligns with his or her goals. For example, if the goal is to maximize detection rate, the RF supervised model is a great choice. However, one may select the Hybrid-RNN to minimize FPR.

Table 6. 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-CNN training	92564.27 ± 1406.58 (5000 samples)
SHIL-RNN training	348281.47 ± 4605.95 (5000 samples)
SHIL-RF 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-CNN detection	7.41 ± 0.001 per sample
SHIL-RNN detection	9.50 ± 0.048 per sample
SHIL-RF detection	7.65 ± 0.11 per sample

Figure 8 summarizes the sensitivity study results across threat impact categories. Similar to the overall results, Figure 8a shows that the SHIL approaches, with outlier detection and similarity filtering, have the lowest false positive rate over their hybrid counterparts. The exceptions occur for the RNN methods and, in general, for the smaller categories, “Crash the application” and “Escalate privilege level". The hybrid RF results tend to be slightly higher than the corresponding hybrid RNN and hybrid CNN results. Overall, the hybrid RNN approaches generally have the lowest FPR result across the threat impact categories. For instance, SHIL-RNN and Hybrid-RNN-OD yield the lowest FPR in all categories but “Disclose credential information” and “Crash the application”. Although the Hybrid-CNN obtains the highest FPR in the “Disclose credential information” and “Consume excessive CPU” categories, SHIL-CNN obtains one of the lowest FPR with outlier detection and filtering.

Figure 8b illustrates that the detection rates of the hybrid approaches are generally stable over outlier detection and filtering status, especially for the hybrid RF alternatives. Nevertheless, SHIL-RF, with both outlier detection and similarity filtering, achieves the best hybrid RF results in all categories but “Disclose credential information”. The RF approaches attain the best detection rates than the other hybrid methods in all categories but “Return a shell and execute arbitrary code” and “Crash the application”. The hybrid CNN methods outperform the others in those two categories. Whereas, the RNN approaches tend to attain the lowest detection rates.

Figure 8c depicts the lead time results. The hybrid configurations for each supervised model achieve similar results across the threat impact groups. In the “Consume excessive CPU”, “Crash the application”, and “Escalate privilege level” categories, adding outlier detection and filtering gives similar or better lead time. Whereas, the opposite trend occurs for the “Disclose credential information” attacks. Overall, the CNN approaches obtain the most cases of the best results, with instances in the “Execute arbitrary code”, “Consume excessive CPU”, and “Escalate privilege level" classes. In contrast, the RNN methods have the most cases of the lowest lead time, especially in the “Execute arbitrary code”, “Disclose credential information”, and “Consume excessive CPU” categories.

System run time measurements. Table 6 compares the training and testing time of CDL, Self-Patch, SHIL-CNN, SHIL-RNN, and SHIL-RF. SHIL-RF takes slightly more time than the unsupervised approaches, CDL and Self Patch, during training and testing due to more processing steps in the system. The SHIL-CNN and SHIL-RNN have the longest training time, given their supervised model complexity. Nevertheless, the training can be improved with graphical processing units (GPUs). For instance, our initial tests with an NVIDIA GeForce RTX 3060 Ti graphics card improve training time by 8.0× for SHIL-CNN and 5.5× for the SHIL-RNN. The learning methods employed by SHIL-RF (autoencoder and random forest) are all light-weight. 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.

In summary, our results show that SHIL can significantly reduce false positive rates while maintaining similar or higher detection rate compared to other unsupervised machine learning approaches. Compared to pure supervised learning approaches, SHIL is able to overcome the low detection rate and high false positive rate by avoiding requiring high-quality training data labels. Furthermore, SHIL can achieve high detection accuracy consistently across different security threat categories. The runtime execution time results show that SHIL is practical for real-time security attack detection in large-scale containerized environments.

However, SHIL has its limitations. First, we observe SHIL has low detection rate with attacks that crash the application and cause data losses. In the "Escalate privilege level" category, SHIL shows higher false positive rate and smaller lead time than its results in the other categories. Finally, SHIL does not focus on real-time model updates so SHIL may suffer from false positives in episodes of rapidly changing workloads. We plan to address those limitations in our future work.

3.3 Case Study

In this subsection, we analyze one representative attack from each threat impact category to provide context about how SHIL identifies attacks with lower false positive rate compared with the alternative learning methods. We show that SHIL better identifies and avoids false positive samples that often result from the dynamic workload frequency vector patterns.

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. When SHIL detects the attack, there are a number of system calls changing in the same sample. The increase in the `epoll_ctl`, `epoll_wait`, `read`, and `write` system calls are the top anomalous changes. Compared with the unsupervised model, SHIL reduces the false positive rate by 88.05% without decreasing the detection rate. In addition, the supervised models have up to 82% higher FPR than SHIL. Whereas, the semi-supervised method suffers from 50% lower detection rate. The false positive samples contain those system calls that occur before the attack is triggered but with fluctuating frequencies. The unsupervised model identifies them as anomalies by mistake. We observe that the `stat` system calls are generated by the dynamic workload as the normal period exhibits similar time-varying patterns. SHIL's self-supervised model can identify the normal dynamic pattern and filter out those false positives via similarity filtering.

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. SHIL detects samples that exhibit more production of certain system calls than usual including `connect`, `mmap`, `sendto`, and `socket`. Compared to the unsupervised model, SHIL reduces false positive rate by 87% with no reduction in detection rate. In addition, the best supervised model only detects one out of the four attack cases with worse FPR than the unsupervised model while the semi-supervised method does not detect any attack

occurrence for this CVE. Similar to the previous attack type, SHIL successfully filters out many false alarms that are caused by dynamic workloads.

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 detects samples showing high production of some system calls like `close`, `fstat`, `mmap`, `mprotect`, `open`, and `read`. SHIL filters out 82 false positives, reducing FPR by 91.1% without hurting the detection rate. In addition, the best supervised model result shows four times the FPR of the unsupervised model with the same detection rate. The semi-supervised method misses one detection case and still has 2.7 times more FPR than SHIL. We observe that certain system calls (`accept`, `stat`, `close`, `fstat`, `read` and `mmap`) vary widely and periodically even across normal run samples, which is challenging for the unsupervised method.

Consume excessive CPU. Apache Commons suffers from an exposure in CVE-2014-0050 that causes excessive CPU usage. The `multipartstream` class of Apache Commons accepts forms with `multipart` contents, separated by a boundary string. The forms are then read into a buffer to accept data chunks along with the boundary between them. A malicious multipartstream request uses an excessively long boundary string such that the buffer is fully filled and no new data can come in. Apache Commons reads data from the buffer using a loop. Since new data cannot be read into the buffer, Apache Commons hangs in the infinite loop. SHIL detects samples containing a large number of the `sched_yield` system calls and numerous other system calls such as `read` and `write`. The unsupervised model detects all four attacks in different containers but with a high FPR of 5.2%. SHIL reduces the FPR by 90% while missing one attack case. The supervised models either have similar high false positive results to the unsupervised method or fail to detect any attack. The semi-supervised method only detects one case and incurs 25% higher FPR than SHIL. SHIL removes 110 false positives. For those false positives, system calls such as `futex`, `lseek`, and `read` have decreased frequencies. The `close`, `fstat`, `open`, and `stat` calls have increased frequencies. The changes occur every four seconds during normal run, and are generated from the workload. SHIL captures these periodic patterns better than the unsupervised anomaly detection.

Crash the application. The Network Time Protocol (NTP) application crashes when CVE-2016-7434 is exploited. The attacker issues a query to the NTP application. Then NTP reads the field values and decodes them from the query, which invokes the `strlen` function to read data from a specific memory location. The malicious user provides a null input so that `strlen` calculates the length of a string with a null address. The null pointer de-reference causes a segmentation fault, which leads to the denial of service. SHIL detects samples with lower system call frequencies than usual from `gettid`, `recvmsg` and `read` calls, but with some new occurrences of `close` and `munmap`. The unsupervised model detects the attack in three cases but with high FPR. SHIL misses one more case but lowers the FPR by 99%. In addition, most supervised and semi-supervised methods only detect one case with little or no lead time before the crash. SHIL filters out 197 false positives, which are generated by the dynamic workload. For example, in certain false positives, `clock_gettime`, `read`, `rt_sigprocmask`, `recvmsg`, `select`, `sendto`, and `write` have higher invoking frequencies, where those system calls are relevant to retrieving time information and processing NTP requests over a socket connection.

Escalate privilege level. CVE-2017-12635 is a Couchdb vulnerability that allows malicious users to gain administrative privileges. Couchdb users can provide multiple roles when creating user accounts for the database. Couchdb uses Erlang-based and JavaScript-based JSON parsers to store the roles but accidentally performs the safety check only on the role that the JavaScript parser accepts to make sure it does not have any unauthorized administrative

access. The attacker provides a first administrator role which is accepted by the Erlang-based parser, and a second non-administrator role which is accepted by the JavaScript parser. Couchdb grants the administrator account creation to the attacker because Couchdb only checks the second role is non-administrative but still authorizes both roles. The malicious user can then act as an administrator using the new account. When SHIL detects the attack, there is an increase in the `fstat`, `mmap`, `mprotect`, and `stat` system calls, compared to normal samples. There is also a large number of `sched_yield` system calls but they also exist in the normal samples. For this CVE, SHIL filters 163 false positive samples, reducing FPR by 80% with no detection rate loss. In addition, most supervised methods have more than 50% higher FPR than SHIL. Although the semi-supervised method has similar FPR as SHIL, its detection rate drops by 25%. We observe `sched_yield` has a higher frequency in certain false positives. In those filtered false positives, we observe a few new occurrences of `mmap` and `munmap` but a larger increase in `sched_yield`. However, the `sched_yield` calls happen periodically during normal execution. SHIL can successfully filter out those false positives.

4 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. [25] studied 11 privilege escalation exploits and proposed a defense mechanism to defeat privilege escalation attacks. Lin et al. considered bugs particularly threatening to container isolation and evaluated Linux hardening techniques like secure computing mode (seccomp) against them. Instead, we take a dynamic analysis approach and focus on vulnerabilities in containerized applications themselves.

Self-Patch [41] combined light-weight dynamic attack detection and targeted patching to achieve effective security protection for containerized applications. In comparison, SHIL improves detection accuracy by providing a new hybrid self-supervised learning approach. CDL [26] was a classified distributed learning framework for containerized applications that proposed application-specific unsupervised detection models. Lindvärn [27] et al. proposed using isolation forest for anomaly detection to achieve good detection rate and a relatively low false positive rate for 22 attacks. Compared to these works [26, 27, 41], we devise a hybrid alert mechanism over samples near the unsupervised anomaly detection threshold with self-supervised models to further reduce false positives.

DIVA [38] performed vulnerability detection on Docker Hub images. Similarly, DIVDS [23] diagnosed Docker images when they are uploaded or downloaded from Docker image repositories. Thus, DIVA and DIVDS are static detection approaches that detect vulnerabilities in container images while SHIL monitors real-time system activity.

SHIL complements the existing container vulnerability detection 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. [33] proposed a decentralized fog-based attack detection framework which uses the semi-supervised fuzzy c-means (SFCM) algorithm with an extreme learning machine (ELM) to detect attacks that occurred on internet of things (IoT) devices. They monitored labeled network data, focusing on attacks that trigger denial-of-service, disclose information, or escalate privilege level at the fog layer. While SHIL monitors the system calls of containerized applications, identifying attacks that execute code and consume excessive CPU in addition.

Idhammad et al. [18] introduced a semi-supervised learning approach for distributed denial-of-service (DDoS) detection based on network entropy estimation, co-clustering, information gain ratio, and extra trees classifiers. Compared to SHIL, the authors used the unsupervised clustering method as a noise filtering step to narrow down

incoming network traffic to those packets important for the decision-making supervised ensemble. Zimba et al. [45] proposed a semi-supervised algorithm based on shared nearest neighbour (SNN) clustering to detect advanced persistent threat (APT) attacks. Unlike SHIL, Zimba et al. focused on subtle APT attacks by tracking a variety of network data including unlabelled DNS logs and network flows in addition to labelled compromised host activity. They used the SNN clustering approach to extract features from unlabelled dataset that will be processed by the KNN classifier. Khonde et al. [21] described an ensemble-based semi-supervised learning approach for a distributed intrusion detection system. Khonde et al. examined network traffic to detect ransomware, DDoS, and escalation attacks. SHIL can also track these categories in system call data. However, they select features based on the Gini index and variable importance measures to run their ensemble of five supervised classifiers.

Compared with the existing work which start 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. [8] described a three-layered system using supervised learning for intrusion detection for smart home IoT devices. SHIL uses supervised models to cross validate the decisions of the unsupervised model while Anthi et al. used supervised methods as the decision making model for network-based attacks such as DoS, man-in-the-middle (MITM)/spoofing, reconnaissance, and replay. Khan et al. [20] proposed an intrusion detection system combining multiple supervised learning methods. The authors used Spark ML to implement traditional classifiers that determine abnormal network packets, combined with a convolutional-LSTM to categorize misuse attack signatures. However, their goal was to identify network misuse attacks given labeled data. DTB-IDS [31] presented a decision-tree-based anomaly detection method to detect APT attacks. Although DTB-IDS used a random forest model like SHIL, DTB-IDS collected behavior information from labeled system, network, and static analysis features to detect APT malware code. DISTDET detects APT attacks in a decentralized manner by monitoring process, file, and network system events using a provenance graph [11]. DISTDET performs lightweight detection on the client side with hierarchical system event tree (HST) models and then applies more intensive false positive filtering techniques on the server side with a ranking algorithm. SHIL uses more lightweight system call frequencies instead of system entities, and does not need labelled training data.

Aksu et al. [6] applied the Fisher Score algorithm to select features and fed the features into the support vector machine (SVM), k-nearest neighbour (k-NN) and decision tree (DT) algorithms for intrusion detection. Aksu et al. also implemented their approach for network data (CICIDS2017 data set) to primarily detect DoS based attacks. Hosseini et al. [17] performed incremental learning with supervised models to detect DDoS attacks. Their work spread its computational load among the client, proxy, and server sides to overcome the resource-constrained scenario of DDoS attacks. SHIL addresses all kinds of security attacks by inspecting unlabelled system call activity.

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 pure unsupervised learning methods to intrusion detection. Unicorn [15] used K-medoids and data provenance analysis to detect APT cyberattacks. Although Unicorn used unsupervised clustering, its labeled provenance graph gave additional context of causally related system events to facilitate detecting long-running APT attacks. The focus of our work is not APT attacks. AIRTAG analyzes DNS, firefox, and syslog data with an unsupervised one-class SVM [10]. However, they

process the logs into causal graphs with pre-trained natural language models. In contrast, SHIL processes system calls into frequency vectors to monitor anomalous system behavior.

Fu et al. propose a network intrusion detection system, pVoxel that aims to lower false alarms in existing ML based traffic detection systems [13]. pVoxel uses point cloud analysis to capture the topological features among the alarm points and leverages k-means clustering algorithm to filter low density false alarm groups. pVoxel is designed for flow-level traffic detection methods and for network attacks such as DDoS attacks and IP spoofing. SHIL detects attacks with system call level monitoring by initially considering alarms with boundary case errors. SHIL does not immediately output the labels from its unsupervised outlier detection method, but further reduces false alarms by using the fine-grained labels to train a validating supervised model. Li et al. propose RETSINA to detect zero-day web attacks using data from different web domains under a business [24]. RETSINA builds an unsupervised universal model that each domain adapts with meta-learning using relatively few domain training data. Nevertheless, RETSINA has adequate data for making the universal (LSTM-based autoencoder) model. In contrast, SHIL uses application-specific models in ephemeral container-based environments. SHIL also processes system calls instead of HTTP requests texts to detect security attacks. ARGUS focuses on detecting situational malicious actions in control-plane apps and services that control IoT devices called contextual attacks [34]. They analyze sensor values and device states with their unsupervised deep autoencoder of gated recurrent unit (GRU) layers. SHIL takes additional cross-validation steps with a supervised model to address false alarms in extensive security attacks scenarios. Scholkopf et al. [37] proposed a one-class support vector machine (SVM) and defined a frontier as a threshold for outlier detection. Liu et al [28] isolated anomalies from normal data by randomly selecting a feature and a value in the possible range to split data points. These two works presented novel outlier detection and anomaly detection methods but do not focus on a security context.

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. [35] 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. Gruhl et al. [14] present mCANDIES to tackle online learning in self-improving systems under continuously changing dynamic data by exploiting the locality and density of data. Such approaches can support detection systems like SHIL by informing when changes to underlying statistical properties of expected and abnormal data warrant model adjustments in practice. Furthermore, Shu et al. [39] 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 underlying program context.

In summary, SHIL presents a new container security attack detection approach that monitors the system level activities of containerized applications with self-supervised machine learning models. SHIL presents a unique hybrid approach that augments the primary unsupervised anomaly detection model with supervised learning models for special cases near the anomaly threshold, differentiating it from semi-supervised methods. SHIL lowers the false alarms of unsupervised anomaly detection methods by obtaining training labels automatically for training effective supervised methods that work in conjunction with unsupervised models.

5 CONCLUSION

In this paper, we have presented SHIL, a new self-supervised hybrid learning framework 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 93% while maintaining similar detection rates compared to existing supervised, unsupervised, or semi-supervised methods. SHIL is light-weight and does not require manual data labelling, which makes it practical for security attack detection in container-based production environments.

6 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] 2017. Docker image vulnerability research. https://www.federacy.com/docker_image_vulnerabilities
[2] 2018. Tesla's cryptojacking attack. https://redlock.io/blog/cryptojacking-tesla
[3] 2021. CVE-2021-44228 Detail. https://nvd.nist.gov/vuln/detail/CVE-2021-44228
[4] 2021. Secure DevOps for Containers, Kubernetes, and Cloud | Sysdig. https://www.sysdig.com
[5] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. 2001. On the surprising behavior of distance metrics in high dimensional space. In *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*. Springer, 420–434.
[6] Doğukan Aksu, Serpil Üstebay, Muhammed Ali Aydin, and Tülin Atmaca. 2018. Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In *International Symposium on Computer and Information Sciences*. Springer, 141–149.
[7] Jinwon An and Sungzoon Cho. 2015. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE* 2, 1 (2015), 1–18.
[8] E. Anthi, L. Williams, M. Słowińska, G. Theodorakopoulos, and P. Burnap. 2019. A Supervised Intrusion Detection System for Smart Home IoT Devices. *IEEE Internet of Things Journal* 6, 5 (2019), 9042–9053. https://doi.org/10.1109/JIOT.2019.2926365
[9] Eric Carter. 2019. Sysdig 2019 Container Usage Report: New Kubernetes and security insights. https://sysdig.com/blog/sysdig-2019-container-usage-report/
[10] Hailun Ding, Juan Zhai, Yuhong Nan, and Shiqing Ma. 2023. AIRTAG: Towards Automated Attack Investigation by Unsupervised Learning with Log Texts. In *32nd USENIX Security Symposium (USENIX Security 23)*. 373–390.
[11] Feng Dong, Liu Wang, Xu Nie, Fei Shao, Haoyu Wang, Ding Li, Xiapu Luo, and Xusheng Xiao. 2023. DISTDET: A Cost-Effective Distributed Cyber Threat Detection System. In *32nd USENIX Security Symposium (USENIX Security 23)*. 6575–6592.
[12] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. 1996. A sense of self for unix processes. In *Proceedings 1996 IEEE Symposium on Security and Privacy*. IEEE, 120–128.
[13] Chuanpu Fu, Qi Li, Ke Xu, and Jianping Wu. 2023. Point Cloud Analysis for ML-Based Malicious Traffic Detection: Reducing Majorities of False Positive Alarms. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1005–1019.
[14] Christian Gruhl, Bernhard Sick, and Sven Tomforde. 2021. Novelty detection in continuously changing environments. *Future Generation Computer Systems* 114 (2021), 138–154.

[15] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. 2020. Unicorn: Runtime provenance-based detector for advanced persistent threats. *arXiv preprint arXiv:2001.01525* (2020).

[16] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[17] Soodeh Hosseini and Mehrdad Azizi. 2019. The hybrid technique for DDoS detection with supervised learning algorithms. *Computer Networks* 158 (2019), 35–45.

[18] Mohamed Idhammad, Karim Afdel, and Mustapha Belouch. 2018. Semi-supervised machine learning approach for DDoS detection. *Applied Intelligence* 48, 10 (2018), 3193–3208.

[19] Justin M Johnson and Taghi M Khoshgoftaar. 2019. Survey on deep learning with class imbalance. *Journal of Big Data* 6, 1 (2019), 1–54.

[20] Muhammad Ashfaq Khan, Md Karim, Yangwoo Kim, et al. 2019. A scalable and hybrid intrusion detection system based on the convolutional-LSTM network. *Symmetry* 11, 4 (2019), 583.

[21] SR Khonde and V Ulagamuthalvi. 2019. Ensemble-based semi-supervised learning approach for a distributed intrusion detection system. *Journal of Cyber Security Technology* 3, 3 (2019), 163–188.

[22] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[23] Soonhong Kwon and Jong-Hyouk Lee. 2020. DIVDS: Docker Image Vulnerability Diagnostic System. *IEEE Access* 8 (2020), 42666–42673.

[24] Peiyang Li, Ye Wang, Qi Li, Zhuotao Liu, Ke Xu, Ju Ren, Zhiying Liu, and Ruilin Lin. 2023. Learning from Limited Heterogeneous Training Data: Meta-Learning for Unsupervised Zero-Day Web Attack Detection across Web Domains. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1020–1034.

[25] Xin Lin, Lingguang Lei, Yuewu Wang, Jiwu Jing, Kun Sun, and Quan Zhou. 2018. A measurement study on linux container security: Attacks and countermeasures. In *Proceedings of the 34th Annual Computer Security Applications Conference*. 418–429.

[26] Yuhang Lin, Olufogorehan Tunde-Onadele, and Xiaohui Gu. 2020. CDL: Classified Distributed Learning for Detecting Security Attacks in Containerized Applications. In *Annual Computer Security Applications Conference* (Austin, USA) *(ACSAC '20)*. Association for Computing Machinery, New York, NY, USA, 179–188.

[27] Marcus Lindvärn and Zack Lundqvist. 2021. Refining Security Monitoring Techniques for Container-Based Virtualisation Environments. (2021).

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 413–422.

[29] Ming Liu, Zhi Xue, Xianghua Xu, Changmin Zhong, and Jinjun Chen. 2018. Host-based intrusion detection system with system calls: Review and future trends. *ACM Computing Surveys (CSUR)* 51, 5 (2018), 1–36.

[30] Check Point Software Technologies Ltd. 2021. https://blog.checkpoint.com/2021/12/13/the-numbers-behind-a-cyber-pandemic-detailed-dive/

[31] Daesung Moon, Hyungjin Im, Ikkyun Kim, and Jong Hyuk Park. 2017. DTB-IDS: an intrusion detection system based on decision tree using behavior analysis for preventing APT attacks. *The Journal of supercomputing* 73, 7 (2017), 2881–2895.

[32] Aaron Newcomb. 2021. Sysdig 2021 container security and usage report: Shifting left is not enough. https://sysdig.com/blog/sysdig-2021-container-security-usage-report/

[33] Shailendra Rathore and Jong Hyuk Park. 2018. Semi-supervised learning based distributed attack detection framework for IoT. *Applied Soft Computing* 72 (2018), 79–89.

[34] Phillip Rieger, Marco Chilese, Reham Mohamed, Markus Miettinen, Hossein Fereidooni, and Ahmad-Reza Sadeghi. 2023. ARGUS: Context-Based Detection of Stealthy IoT Infiltration Attacks. , Article 241 (2023), 18 pages.

[35] Ishai Rosenberg and Ehud Gudes. 2017. Bypassing system calls–based intrusion detection systems. *Concurrency and Computation: Practice and Experience* 29, 16 (2017), e4023.

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation.* Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[37] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. 2000. Support vector method for novelty detection. In *Advances in neural information processing systems*. 582–588.

[38] Rui Shu, Xiaohui Gu, and William Enck. 2017. A study of security vulnerabilities on Docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*. 269–280.

[39] Xiaokui Shu, Danfeng Daphne Yao, and Barbara G Ryder. 2015. A formal framework for program anomaly detection. In *International Symposium on Recent Advances in Intrusion Detection*. Springer, 270–292.

[40] Olufogorehan Tunde-Onadele, Jingzhu He, Ting Dai, and Xiaohui Gu. 2019. A study on container vulnerability exploit detection. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 121–127.

[41] Olufogorehan Tunde-Onadele, Yuhang Lin, Jingzhu He, and Xiaohui Gu. 2020. Self-Patch: Beyond Patch Tuesday for Containerized Applications. In *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 21–27.

[42] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. 2017. Applying convolutional neural network for network intrusion detection. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 1222–1228.

[43] James Wetter and Nicky Ringland. 2021. Understanding the impact of Apache Log4j vulnerability. https://security.googleblog.com/2021/12/understanding-impact-of-apache-log4j.html

[44] David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*. 189–196.

[45] Aaron Zimba, Hongsong Chen, Zhaoshun Wang, and Mumbi Chishimba. 2020. 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* 106 (2020), 501–517.