# AppMine: **Behavioral Analytics for Web Application Vulnerability Detection**

Indranil Jana and Alina Oprea
Khoury College of Computer Sciences, Northeastern University, Boston, MA, USA

## ABSTRACT

Web applications in widespread use have always been the target of large-scale attacks, leading to massive disruption of services and financial loss, as in the Equifax data breach. It has become common practice to deploy web applications in containers like Docker for better portability and ease of deployment. We design a system called AppMine for lightweight monitoring of web applications running in Docker containers and detection of unknown web vulnerabilities. AppMine is an unsupervised learning system, trained only on legitimate workloads of web applications, to detect anomalies based on either traditional models (PCA and one-class SVM), or more advanced neural-network architectures (LSTM). In our evaluation, we demonstrate that the neural network model outperforms more traditional methods on a range of web applications and recreated exploits. For instance, AppMine achieves average AUC scores as high as 0.97 for the Apache Struts application (with the CVE-2017-5638 exploit used in the Equifax breach), while the AUC scores for PCA and one-class SVM are 0.81 and 0.83, respectively.

**ACM Reference Format:**
Indranil Jana and Alina Oprea. 2019. AppMine: Behavioral Analytics for Web Application Vulnerability Detection. In *2019 Cloud Computing Security Workshop (CCSW'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3338466.3358923

## 1 INTRODUCTION

Web-based attacks remain one of the major attack vectors with notorious security incidents such as the Equifax breach and Drupalgeddon being attributed to web application vulnerabilities [27]. Such attacks already resulted in serious breaches of confidential and personal information affecting consumers and businesses alike. For instance, the Equifax security incident from 2017 impacted approximately 143 million U.S. consumers, compromising their identity (e.g., SSN, driver license) and financial information (e.g., credit card numbers) [14]. We thus expect that cyber security attacks will induce even more devastating consequences in the future.

Containers have gained increased adoption recently for deploying web applications in public and private cloud environments [38]. While containers offer flexibility, scalability, and have low performance overhead, on the downside existing defenses for enterprise networks (e.g., anti-virus, intrusion prevention systems, web proxies) are not readily applicable in container-based environments. Existing solutions based on static analysis, dynamic analysis, input validation, and fuzz testing have well-known limitations.

In this paper we design an unsupervised learning framework called AppMine for web application defense that is effective at protecting container-based applications against a range of vulnerabilities. Our framework monitors web applications deployed in Docker containers and collects detailed run-time information using well-known monitoring tools such as Sysdig [48]. At the core of the technical approach lies a machine learning framework that uses Recurrent Neural Network (RNN) architectures popular in the deep learning community for representing sequential dependencies in time-series data. We use a type of RNN called Long-Short Term Memory (LSTM), which can capture the time evolution of application profiles, learn both short-term and long-term dependencies, and identify security anomalies that deviate from historical behavior.

To evaluate our techniques, we deploy a testbed in which we set up four popular web applications in Docker containers, and recreate seven exploits, using Metasploit modules. We collect system call data using the Sysdig monitoring agent. We compare our LSTM model with two traditional anomaly detection algorithms: Principal Component Analysis (PCA) and One-Class Support Vector Machines (OCSVM). We demonstrate the effectiveness of our framework at detecting well-known vulnerabilities such as those in the Apache Struts and Drupal applications with low false positive rates. Our LSTM model significantly improves upon more traditional anomaly detection models, by exploiting the temporal relationship between application system calls. For instance, for the Apache Struts web application with the CVE-2017-5638 exploit used in the Equifax breach, the LSTM model achieves an AUC of 0.97, while PCA and OCSVM have AUCs of 0.81 and 0.83, respectively. The advantages of our AppMine framework are that it does not require attack data for training, and it can be applied to detect unknown vulnerabilities in web applications.

To summarize, our contributions are:

(1) We design an unsupervised learning framework called AppMine for detection of a range of web application vulnerabilities.
(2) We set up a testbed with four web applications and recreate seven exploits, collecting monitoring data from Sysdig agents deployed in Docker containers.
(3) We compare the performance of our neural-network LSTM model with that of traditional unsupervised learning models (PCA and OCSVM) and demonstrate its effectiveness.

*Organization.* We provide background on web application vulnerabilities and our threat model in Section 2. We describe our methodology, testbed setup, data collection, and machine learning framework in Section 3. We evaluate our system AppMine on


*CCSW'19, November 11, 2019, London, United Kingdom*

ACM ISBN 978-1-4503-6826-1/19/11…$15.00
https://doi.org/10.1145/3338466.3358923

a range of web applications and vulnerabilities in Section 4. We present related work in Section 6 and conclude in Section 7.

## 2 BACKGROUND AND OVERVIEW

We first provide some background on web application vulnerabilities, then discuss our threat model, and give an overview of our system AppMine.

### 2.1 Web application vulnerabilities

Web applications can be deployed either on-premise (in private clouds and data centers) or off-premise (in public cloud environments). A recent trend is to *containerize* web applications, which involves running them in Docker containers for increased portability across different hardware and software stacks. Docker adoption for public cloud customers has increased to 49% in 2018, compared to 35% a year before, according to a report from RightScale [38]. Also, container-as-a-service platforms are experiencing increased adoption (44% adoption of Amazon's ECS/EKS service in 2018 [38]). It has been reported that two thirds out of a set of 576 surveyed IT leaders plan to migrate from VMs to container use in public clouds [10].

Vulnerabilities in web and database applications might expose enterprise networks to serious security breaches. For instance, several remote code execution vulnerabilities (CVE-18-9805 and CVE-18-11775) have been discovered for Apache Struts, a Java open source framework for developing web applications. Among these, the famous Apache Struts vulnerability CVE-2017-5638 allows remote attackers to execute arbitrary commands on the web server via the Content Type HTTP header. After getting access to the web server, attackers start propagating laterally in the network and reach the target of interest (usually a database storing confidential information). Other web vulnerabilities include SQL injection, cross-site scripting, and cross-site request forgery.

Defenses against these application vulnerabilities usually involve patching vulnerable applications, after the exploit is known and a patch is available. Web application security testing tools use a combination of static code analysis [21], dynamic checks [49], input validation [43], and fuzz testing [42], but in general they have a number of well-known limitations (e.g., high false positives or false negatives). In general, despite all existing defenses, enterprises are still exposed against zero-day vulnerabilities in their web applications.

### 2.2 Threat model

We consider a system model in which web applications run in container environments such as Docker and monitoring agents are deployed in the containers. We assume that the containers themselves and the monitoring agents that collect the monitoring data are not under the attacker's control. Attackers are performing their actions remotely, interacting with the web application via network packets. We also assume that the adversary cannot tamper with the collected system call data. Attackers with access to the monitoring environment and the system logs are much more powerful, and are beyond our current scope.

We thus handle scenarios in which attackers interact remotely with the web application, attempting to exploit a vulnerability. The web application attack is usually an entry point for the attacker, interested in moving laterally in the environment (cloud or enterprise) and obtaining access to critical resources. For instance, in the Equifax breach, the Struts exploit allowed the attacker control of the web server, but the attacker's ultimate goal was obtaining access to the database containing personal information of millions of customers.

### 2.3 Overview

In designing our machine learning framework for web application threat detection, we leverage several main insights. First, there is a large amount of sequential dependence in application behavior manifested in unique, highly distinguishable sequences of system calls generated by an application. We argue that machine-learning techniques applied to application monitoring independently of their temporal ordering and contextual information do not offer sufficient protection against advanced attacks. Therefore, we design machine learning architectures that leverage the sequential, temporal dependencies in application monitoring data. Second, we believe that application exploits and cyber attacks will result in deviation of monitoring data compared to an application running under normal conditions without exposure to the attack or vulnerability. To the extent the attack is observable in the collected monitoring data depends on a number of factors including the attack specifics and the granularity of collected data. Third, by encapsulating single applications in each container, we can reduce the amount of noise from events generated by other applications, and create "clean profiles" of typical enterprise applications.

A number of challenges need to be addressed in designing our machine learning-based application threat detection framework. We highlight among them: (1) privacy and performance considerations that prevent full-blown application monitoring; (2) dealing with non-deterministic application behavior; (3) designing the most appropriate machine learning models for this setting. There is a lot of research and guidance on designing neural network architectures for other domains (e.g., image classification, speech recognition, machine translation), but there is no widely accepted methodology for security datasets that exhibit different semantics.

Figure 1 provides an overview of our machine learning framework including the following components: (1) **Testbed setup** to run various web applications and emulate legitimate behavior; (2) **Vulnerability exploitation** to reproduce existing vulnerabilities using Metasploit; (3) **Data collection** using the Sysdig monitoring agent installed in the Docker container environment; (4) **Machine Learning Anomaly Detection Framework** supporting both traditional and deep-learning based models. We detail each of these components in the next section.

## 3 METHODOLOGY

In this section we discuss our system methodology. We start by describing the testbed setup, the vulnerability exploitation method, and the data collection. Finally, we describe in detail our unsupervised machine learning framework for web application vulnerability detection.

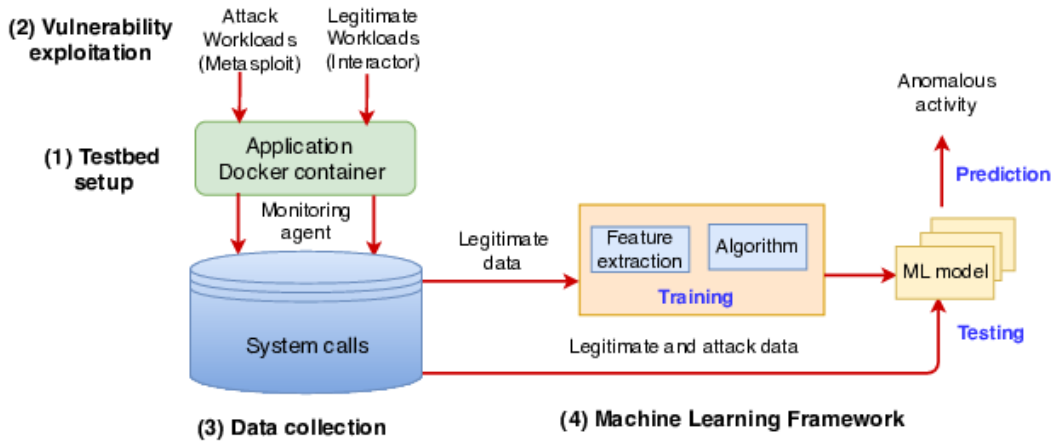



**Figure 1: Overview of the** AppMine **system architecture.**

### 3.1 Testbed setup

Cloud providers that deploy containerized web applications might leverage existing monitoring tools and obtain data from realistic user interactions with the web applications. However, potential privacy implications prevent these providers to perform detailed monitoring and release their datasets to the broader community. As we do not have access to datasets collected from deployed web applications, we create our own monitoring environment for applications deployed in Docker containers. We set up an Ubuntu 16.04 Virtual Machine running Docker and install the Sysdig monitoring agent. For each considered web application, we built and deployed Docker containers on the Ubuntu machine. We used a separate Kali Linux 2 based Virtual Machine to run the scripts for generating the attack data using Metasploit. We discuss now in more details the choice of web applications, how we generate realistic legitimate workloads, and how we set up the exploit emulation.

*Web applications.* The web applications we selected include: Apache Struts [27] (two different variants), Drupal [11], WordPress[51] (three different plugins [52–54]), and ProFTPD [37]. All of these are popular, open-source web applications that experienced vulnerabilities in the last few years.

To select exploits of interest for these applications, we evaluated recent exploits from the CVE database and identified a set of seven exploits that had Metasploit modules. Table 1 details the application description and vulnerabilities that we emulate in our environment. We include CVE numbers for five out of the seven exploits, as the other two do not yet have a CVE number assigned. The CVE-2017-5638 for Apache Struts is the famous remote code execution vulnerability that led to the Equifax data breach.

*Legitimate workloads.* One of the main challenges in our work is setting up workloads and interactions with the web applications that are similar to those generated by actual users. For this purpose, we found traditional web crawling to be unsuitable, as it focuses more on gathering data in an automatic manner than on creating realistic, human-like interaction. Thus, we designed **Interactor**, a program that uses Selenium [44] to realistically interact with elements on the web interface of the web application. **Interactor** opens a browser instance per user and interacts with the page by filling available forms, clicking hyperlinks, and pressing buttons in a randomized order. Small, random delays of 1-5 seconds are also inserted during the operation of **Interactor**, so it simulates user behavior more realistically than a web scraper. Data collection for Struts, Drupal and Wordpress was done using **Interactor**. For ProFTPD, we choose to leverage **ftpbench** [16] for data generation, a benchmark tool that provides login to an FTP sever and file uploading capabilities.

### 3.2 Vulnerability exploitation

The Metasploit Framework [31] is an open source project that serves as a penetration testing platform for finding and exploiting vulnerabilities. For our work, we used existing Metasploit modules for the tested vulnerabilities and modified them as necessary to work on our versions of the web applications. Additionally, Metasploit comes with a set of post-exploitation information gathering modules that can be run on compromised machines, and we used them after exploiting each web application to ensure that we gained access to the victim. This step replicates a likely scenario in real-world exploitation, where information gathering is often the first step an attacker takes after compromising a system, so as to learn the victim's system and network configuration and discover more vulnerabilities. Table 2 lists the post-exploitation scripts we tested and collected data for. They include gather network information, collect system information, user list, and credentials, and dump password hash files.

### 3.3 Data collection

We use Sysdig [48] to monitor the activity of the web applications running in Docker containers. Sysdig has the ability to collect sequences of system calls made by the application in the container.

For each application except ProFTPD, we ran our **Interactor** program to emulate 1, 3, 5, 10 and 15 simultaneous users. Each user performed a number of actions chosen randomly between

| App Name | Description | Plugin Name | CVE |
|---|---|---|---|
| Struts | Framework for developing Java EE webapps | - | CVE-2017-5638 |
| | | | CVE-2017-9805 |
| Drupal | Content-management framework | - | CVE-2018-7600 |
| WordPress | Content management system | Reflex Gallery Plugin | CVE-2015-4133 |
| | | Ajax Load More Plugin | - |
| | | N-Media Website Contact Form | - |
| ProFTPD | FTP server | - | CVE-2015-3306 |

**Table 1: Open-source web applications and vulnerabilities considered for this work.**

| Post-exploit script name | Description |
|---|---|
| checkcontainer | Check whether target running inside container |
| ecryptfs_creds | Collect all users' .ecrypts directories |
| enum_configs | Collect configuration files for Apache, MySQL, etc. |
| enum_network | Gather network information |
| enum_protections | Find antivirus/IDS/firewalls etc |
| enum_psk | Collect 802-11-Wireless-Security credentials |
| enum_system | Gather system information (e.g., installed packages) |
| enum_users_history | Gather user list, bash history, vim history, etc. |
| enum_xchat | Gather XChat's configuration files |
| env | Collect environment variables |
| gnome_commander_creds | Collect cleartext passwords from Gnome-commander |
| hashdump | Dump password hashes for all users |
| mount_cifs_creds | Obtain mount.cifs/mount.smbfs credentials from /etc/fstab |
| pptpd_chap_secrets | Collect PPTP VPN information |
| tor_hiddenservices | Collect TOR Hidden Services hostnames and private keys |

**Table 2: Metasploit post-exploitation scripts used for generating attack data.**

50 and 100, where an action is an activity such as clicking on a hyperlink, button on the web page, or filling a form. Monitoring was started after running the containers, and ended when the last user had performed his last action. For ProFTPD, we used the login and upload benchmarks from ftpbench [16] for 30 mins to 1 hour each to generate legitimate data. For attack data, Sysdig monitoring and data collection was done similarly.

Table 3 shows the data statistics for the four applications we monitored. In total, we collected between 300 and 360 minutes of data for each applications and vulnerability. We split the sessions into training and testing for the machine learning framework, to minimize correlation and ensure independence of training and testing data. In Figure 2, we show the distribution of the top 20 system calls during legitimate use for the WordPress application over a duration of approximately 15 seconds. The attack is being performed against the N-media contact form plugin, and the post-exploit script being run is enum_network. Clearly, the system calls during the attack script are noticeably distinguishable from the ones used during the regular application runs. Notably, some system calls (e.g., fcntl, close) are used more frequently during the attack.

## 3.4 Machine Learning Framework

ML methods for web attack detection generally fall into two categories: supervised and unsupervised methods. Supervised learning methods (for example [6, 7, 28, 45]) train a classifier that relies on labeled attack data at training time, and predict web attacks at testing time. Unsupervised learning methods (see, for example [9, 22, 23, 40, 47]) tend to be more general in the requirement of utilizing only legitimate data for training. These methods learn a model of application behavior at training time using only legitimate data (from one class) and predict anomalies at testing time. However, it is well known that unsupervised learning techniques in security typically generate large amounts of false positives and are challenging to tune in operational settings [46].

Our design choice for AppMine is to use an unsupervised learning framework for web vulnerability detection. Our motivation includes mainly the generality power of unsupervised learning. Our goals for AppMine is to learn the normal behavior of each application (independently of the vulnerabilities known at training time), and determine at testing time the anomalies that are indicative of web vulnerabilities. Importantly, AppMine has the ability to detect *new, unseen vulnerabilities* if they generate anomalous behavior in application system calls. One important consideration for AppMine is to learn the distribution of *temporal sequences of system calls*, as there is much stronger signal of application behavior in sequences of system calls, compared to individual system calls.

*Training and testing the models.* As shown in Figure 1, AppMine leverages only legitimate application data for training a machine

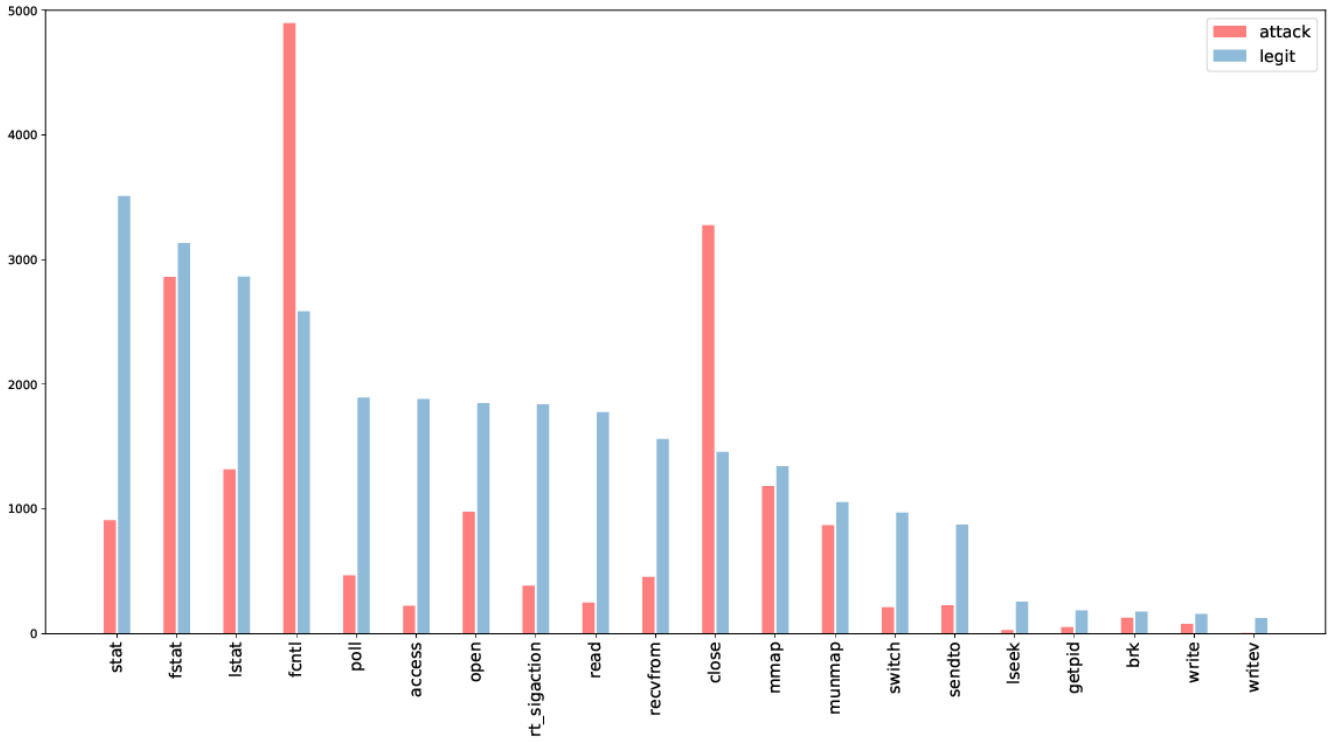



**Figure 2: System call distribution for 15 seconds of data for WordPress, attack being shown in red is the enum_network script**

| Application / Plugin Name | Training data | Legitimate testing data |
|---|---|---|
| Struts CVE-2017-5638 | 282.42 mins | 13.46 mins |
| Struts CVE-2017-9805 | 308.78 mins | 28 mins |
| Drupal CVE-2018-7600 | 301.76 mins | 17.45 mins |
| WP Reflex Gallery Plugin | 323.15 mins | 16.54 mins |
| WP Ajax Load More Plugin | 317.27 mins | 22.42 mins |
| WP N-Media Website Contact Form | 321.05 mins | 18.64 mins |
| ProFTPD CVE-2015-3306 | 342.85 mins | 20 mins |

**Table 3: Amount of data used for training and validation.**

learning model that learns the system call distribution under normal conditions. As already mentioned, we vary the number of users and their actions to create a set of diverse legitimate workloads for training the models. AppMine creates system-call based feature representations for time windows of fixed length. Depending on the ML technique, the feature vector for either one time interval or a sequence of time intervals are given as input to the ML algorithm. The output of the training phase is a model that can determine the likelihood of a certain sequence of system calls. Importantly, as each application exhibits different behavior, we train a model per application to learn the application normal behavior.

At testing time, the application model is applied to new data generated by the same web application. We run the model on both legitimate and attack data for that application. The model produces an *anomaly score* for the feature vectors at a particular time window, indicating the likelihood that the application has been exploited in that time interval. Based on the labeled ground truth, we compute standard metrics such as True Positives, False Positives, and Area Under the Curve (AUC). A graphical representation of our anomaly detection models is given in Figure 3.

*Traditional anomaly detection techniques.* As mentioned, Sysdig collects time-stamped sequences of system calls for web applications. The first question we had to answer was how to represent this data for use in an anomaly detection system. A simple and fairly common representation (see, for example, [13]) is to create a feature vector $x^t = [f_1^t, \ldots, f_s^t]$ storing the frequency for each system call during a fixed time interval $t$. Here $s$ is the total number of system calls and $f_i^t$ denotes the number of times the $i$-th system call has been used during time window $t$. The interval length is a hyper-parameter of the system. System calls that did not appear in

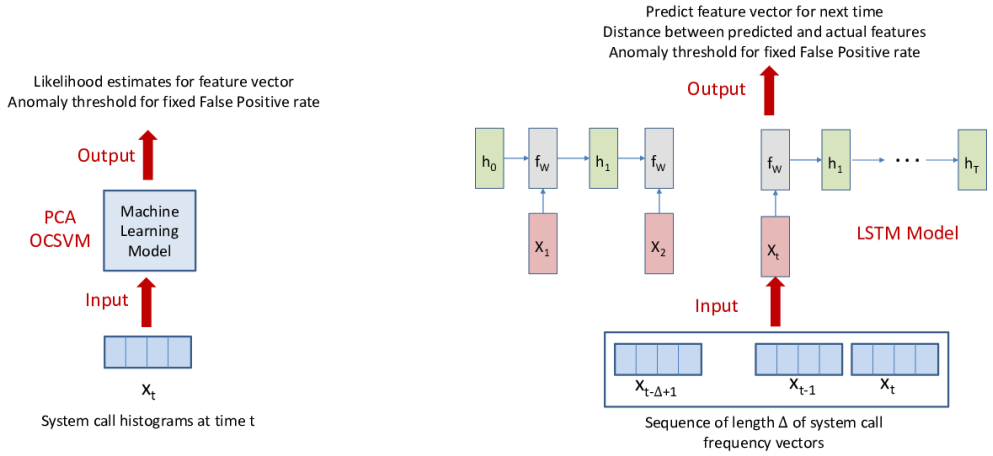



**Figure 3: Machine Learning methods in** AppMine**. Traditional models on the left and the LSTM model on the right.**

that interval have a frequency value of 0. For intervals where no system calls were captured by Sysdig, a feature vector of all zeros was used. For example, let us assume that a program makes the following system calls during a one-second interval:

`futex,futex,open,write,close,open,read,close`

Then, the feature vector for that interval is (system call `exec` is not used and has value 0)

| close | futex | open | write | read | exec |
|---|---|---|---|---|---|
| 2 | 2 | 2 | 1 | 1 | 0 |

We use two standard anomaly detection techniques: Principal Component Analysis (PCA) and One Class Support Vector Machine (OCSVM) based on this frequency feature representation. Both of these methods take as input the system call frequency feature vector computed for one time interval. We use these methods to create baselines for comparison with our neural-network based anomaly detection model that leverages the sequence dependencies among system calls.

**PCA:** Principal Component Analysis (PCA) is a technique that projects a dataset onto a set of *principal components*, determined to maximize the variance observed in the training data. If the original data is given by matrix $X$ (with $n$ rows and $d$ columns, where $n$ is the number of data records in $X$ and $d$ is the size of the feature space), then PCA determines the top $k$ eigenvectors of the co-variance matrix $\Sigma = X^T X$. Let the matrix of the top $k$ eigenvectors be $W_k$ (storing one eigenvector per column). Then the projection of a point $x$ onto the space generated by the top $k$ principal components is given by $\hat{x} = W_k^T x$. The principal components have the property of minimizing the total reconstruction error. PCA can also be trained as a density estimation model, which estimates a probability distribution $p(x)$ over $x \in X$ (in our case for frequency feature vectors), based on Maximum Likelihood estimation [8]. We use this density-estimation variant of PCA. In training we compute Gaussian estimates of the probabilities of system call frequency vectors, while in testing we compute log likelihoods for both attack and legitimate data. If a point has very low log likelihood in testing, then we consider it as an attack point. We vary the threshold for log likelihood in order to obtain ROC curves that evaluate True Positives as a function of fixed False Positive Rates.

**OCSVM:** A Support Vector Machine (SVM) is a supervised learning model that is trained on data points from two different classes, in an attempt to find a separating hyperplane that separates the classes by the maximal margin. One-class SVM is trained on data from only one class (in our case, the legitimate class) and is used to detect novelties (or anomalies) based on the learned representation. We learn a one-class SVM model based on the legitimate training data, and use it to predict scores for testing data points (both attack and legitimate). In this setting, the scores are the distances to the separating decision boundary. Again, we vary the scores for which we predict attacks, to compute True Positives at fixed False Positive Rates.

*Motivation for LSTM models.* System calls generated by an application are temporally correlated. For instance, we observed that the Apache Struts application makes the following sequence of system calls: {`openat`, `getdents`, `close`}, repeatedly. Once we observe the sequence {`openat`, `getdents`}, the probability that the next system call is `close` under normal conditions is extremely high. However, the probability of the next system call being `exec` (as might happen during an attack) is very low. A traditional anomaly detection model (such as PCA or OCSVM) does not have the ability to predict the likelihood of the `exec` system call in the context given by the previous system calls.

To capture these types of sequential dependencies in application monitoring data, we leverage Recurrent Neural Network (RNN) architectures. RNNs are deep learning models designed to map sequential and time series data to sequential outputs [41]. RNNs have been successfully applied for tasks such as machine translation, natural language processing, and time series analysis. Long Short-Term Memory (LSTM) networks [19] are special types of RNNs that introduce special forget gates to better control how information propagates through the network. They maintain a hidden state

$h_t$ at time $t$ that is updated using the current input: $(h_t, x'_{t+1}) = f_W(h_{t-1}, x_t)$. Here $W$ are the model weights, $f_W$ is the function applied inside the neural network (including the linear operation and activation function), $x_t$ is the input to the network at time $t$, and $x'_{t+1}$ is the prediction of feature vector at time $t+1$.

*LSTM model design.* We designed an LSTM architecture that learns a model $f_W$ mapping *sequences of system call frequency vectors* to predictions of the next feature vector. The alternative to this design is to use the sequences of system calls directly, without aggregating them into frequency vectors. However, applications use multiple threads and system calls might arrive slightly out-of-oder when the same code is executed. Thus, we expect the sequences of system call frequency vectors (aggregated over relatively small time intervals) to be much more stable than the sequences of individual system calls.

We thus use the following feature representation for the LSTM model. Given a hyper-parameter $\Delta$ (the sequence size), the input to the LSTM model at time $t$ is the sequence of system call frequency vectors for the last $\Delta$ time windows: $x_t, x_{t-1}, \ldots, x_{t-\Delta+1}$. We train the LSTM model in a supervised manner to predict the next system call frequency vector at time $t+1$, given the true value $x_{t+1}$. Thus, the sequence $x_t, x_{t-1}, \ldots, x_{t-\Delta+1}$ and the prediction $x_{t+1}$ is given as input to the LSTM model. At testing time, the LSTM model learns to predict the next frequency vector: $y'_{t+1} \leftarrow f_W(y_t, \ldots, y_{t-\Delta+1}$ given the observed sequence of length $\Delta$ at time $t$.

One important consideration is how to leverage the supervised LSTM model that predicts the next system call frequency vector to distinguish attack patterns from legitimate profiles. The intuition is that, during training, we have information about the true value of the system call frequency vector $x_{t+1}$ at time $t+1$ and we can measure the distance between the predicted value $x'_{t+1}$ and the actual value $x_{t+1}$. We learn the distribution of the distances observed in training and set a distance threshold to obtain a fixed false positive rate (per application). This is equivalent to picking a threshold $T$ such as the probability $P[||x'_{t+1} - x_{t+1}|| > T] <= p$, for some fixed false positive rate $p$.

During testing, we run the LSTM model on new data $y_1, \ldots, y_t, \ldots$ (including legitimate and attack vectors), and predict at each time interval the frequency vector $y'_{t+1} = f_W(y_t, \ldots, F_{t-\Delta+1})$ based on the previous $\Delta$ frequency vectors: $y_t, \ldots, y_{t-\Delta+1}$. We consider the frequency vector at time $t$ an anomaly if the distance is higher than the threshold computed in training: $||y'_{t+1} - y_{t+1}|| > T$.

The last item we need to discuss is how to measure the distance between the predicted system call frequency vector and the actual one. We found that using a standard distance metric (such as Euclidean distance) $||x_{t+1} - x'_{t+1}||_2$ is not effective. Instead, we leverage ideas from Term Frequency - Inverse Document Frequency (TF-IDF) in information retrieval and we weight each system call by its inverse frequency when defining a new distance between the predicted and actual values: $||x'_{t+1} - x_{t+1}||_S = \sqrt{\sum_{i=1}^{d} \bar{f}_i (x_i^{t+1} - x_i'^{t+1})^2}$, where $d$ is the number of system calls, and $\bar{f}_i$ is the inverse-frequency of the $i^{th}$ system call as seen during training. This method ensures that differences to system calls utilized rarely contribute more to the distance compared to common system calls (those used with high frequency).

# 4 EXPERIMENTAL EVALUATION

In this section, we evaluate our three anomaly detection models, using metrics such as True Positives at fixed False Positive rates, and Area Under the Curve (AUC). We first discuss hyper-parameter choice, then compare the performance of LSTM with that of the traditional ML models (PCA and OCSVM), and finally we present detailed attack detection results per attack script.

*Configuring ML models.* We describe here some of the configuration options and hyper-parameter selection for our models.

For PCA we need to select the number of components. For all of the tested web applications, the cumulative variance graph started to flatten out at around 20 principal components. Figure 4 shows an example for Struts (other applications being similar). Thus, we choose to represent the data using 20 principal components. PCA assigns scores to each data point based on how far the projection of the data point on the lower dimensional space lies from the principal components. Normal, benign data points are expected to have a low score, while anomalous points will have higher scores.

As described in 3.4, we trained our models on frequency count vectors. We tested time interval lengths of 100ms, 500ms, 1s, and 2s for creating feature vectors, and choose to work with **one second** for all models, as for the other options the results were worse (the ROC curves were closer to the diagonal).

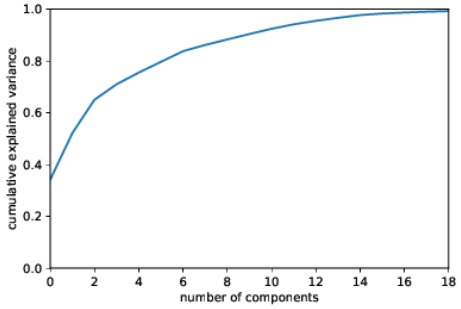



**Figure 4: Cumulative explained variance for PCA for Struts with CVE-2017-5638.**

For training the LSTM models we used Python with Keras and Tensorflow to build a Sequential model with one LSTM layer with 100 hidden neurons and one Dense layer. We used Mean Squared Error (MSE) as the loss function with the ADAM optimization algorithm. Batch size of 128 was used while training over 150 epochs, with a 20% validation split. Our hyper-parameters are given in Table 5. We used several values for the hyper-parameters $\Delta$ denoting the sequence size, once we aggregate the system call frequency vectors at one-second intervals. We show the results for the AUC metric and performance (training time) in Table 4. Based on this, we selected $\Delta = 15$ as the value that offers the best tradeoff between accuracy and performance.

To select a threshold for detecting anomalies, we inspected the distribution of anomaly scores for the frequency vectors in our validation set (containing only legitimate data). In Figure 5 we show the distribution of the distance between predicted values and actual

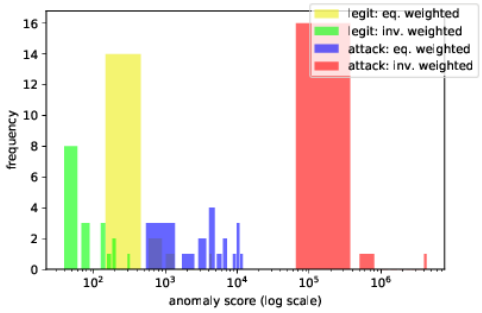



**Figure 5: Distribution of distances between predicted and actual frequency values for the** enum_configs **attack script for Struts with CVE-2017-5638.**

| | $\Delta = 5$ | $\Delta = 10$ | $\Delta = 15$ | $\Delta = 20$ | $\Delta = 30$ | $\Delta = 60$ |
|---|---|---|---|---|---|---|
| AUC | 0.95 | 0.96 | 0.97 | 0.97 | 0.97 | 0.97 |
| Time | 1:46 | 2:56 | 4:30 | 5:27 | 7:15 | 15.05 |

**Table 4: AUC and training time (in minutes) for LSTM with different values of sequence size $\Delta$, for frequency vectors aggregated at one-second intervals (Struts with CVE-2017-5638).**

| Parameter | Values |
|---|---|
| Number of hidden layers | 1 |
| Number of neurons | 100 |
| Batch size | 128 |
| Number of epochs | 150 |
| Timing window | 0.1s, 0.5s, **1s**, 2s |
| Sequence size | 5, 10, **15**, 20, 30, 60 |

**Table 5: Hyper-parameters for LSTM model. Bolded values provide best results.**

frequency values for the Struts application with the enum_config script. We show the distance for attack and legitimate system calls,

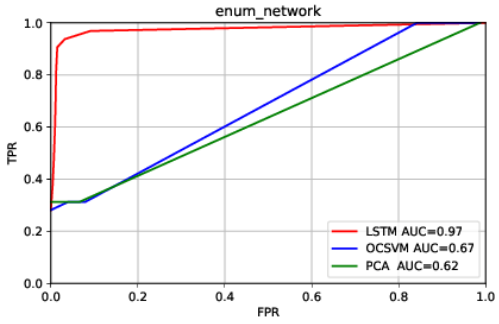



**Figure 6: ROC curves for** enum_system **for Struts with CVE-2017-5638.**

when using either uniform weights, or TF-IDF weights for distance computation. As observed, when using uniform weights the two distance distribution are closer and overlap in some cases. However, with TF-IDF weights, the separation between the two distance distributions increases, making the attack data much more distinguishable from the legitimate one. Thus, we can select a threshold per application to minimize the False Positive rate during training.

*Comparison of LSTM with traditional models.* To compare LSTM with the two traditional anomaly detection models (PCA and OCSVM), Figure 6 shows the ROC curves for one application (Struts) for the attack script enum_network. We observe that LSTM is performing significantly better, with AUC at 0.97, compared to 0.62 for PCA and 0.67 for OCSVM. In the Appendix, we show in Figure 10 the ROC curves for all attack scripts for the Struts application. Interestingly, LSTM is always outperforming the traditional models, with significant increase in AUC. For instance, for the ecryptfs_cred script, LSTM achieves an AUC of 0.96, while PCA and OCSVM have AUCs of only 0.56 and 0.63, respectively.

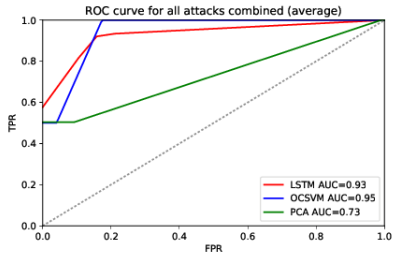
Figure 7 shows the ROC curves averaged on all 15 attack scripts for the all the tested web applications. It is clear that the LSTM model performs better than the traditional models. The AUC for LSTM is between 0.75 and 0.97 and improves the traditional models' average AUC between 0.09 and 0.25.

In terms of the performance of the ML models, LSTM has a more complex architecture and it takes longer to train, as expected. The traditional models' running time is very low, with 0.088 seconds for PCA and 4.37 seconds for OCSVM, respectively (for the Struts 5638 web application). The LSTM model's performance depends on the size of the training data, ranging between 4:30 minutes for Struts 5638 and 5:24 minutes for Word Press.

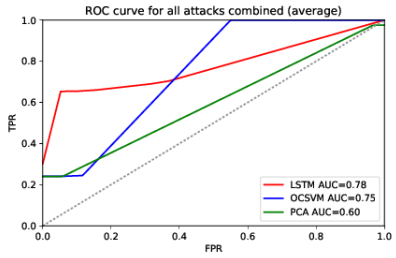
*LSTM for different attacks.* Figures 8 and 9 show the ROC curves for each attack for the Struts and Drupal applications, respectively. The LSTM model generally performs better than PCA or OCSVM There are a few attack scripts that perform poorly for the traditional models (e.g., ecryptfs_creds, enum_configs, and enum_network), but the LSTM models perform much better. We suspect that the reason is the ability of LSTM to analyze sequences of system call frequency vectors. By looking at individual feature vectors for one time window, the patterns of attack and legitimate data might be very similar, but the sequences over multiple time windows could differ significantly.

## 5 DISCUSSION AND LIMITATIONS

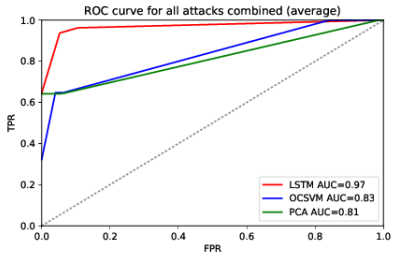
We demonstrated that AppMine is effective at detecting a range of vulnerabilities against four popular web applications, using only system calls collected from Sysdig. There might be situations in which AppMine misses certain classes of attacks, not manifested in the system call sequences generated by an application. For instance, a SQL injection attack might not result in an anomaly at the system call level, but rather in the parameter values used in HTTP requests. We believe that for such attacks, more specific detectors need to be designed, using additional information and features to enhance detection. For instance, SQL detection could leverage parameter values extracted from HTTP requests, relying on network-level features to complement host-level ones.



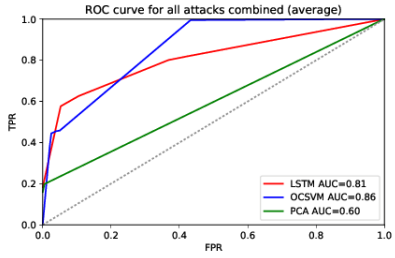

(a) Drupal



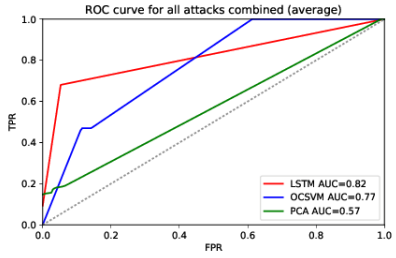

(b) ProFTPD



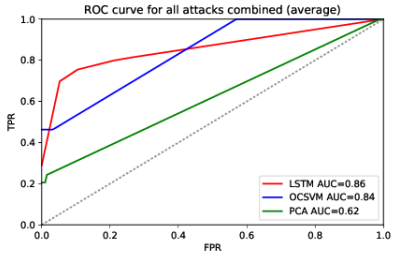

(c) Struts CVE-2017-5638



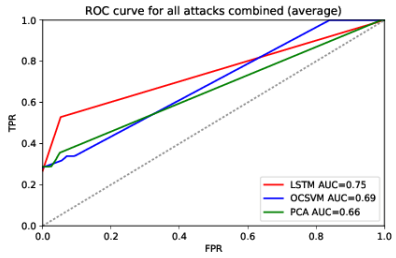

(d) WP Ajax Load More Plugin




(e) WP N-media Contact Form Plugin




(f) WP ReflexGallery Plugin




(g) Struts CVE-2017-9805

**Figure 7: ROC curves for all 15 attack scripts (averaged) for PCA, OCSVM and LSTM for all applications.**

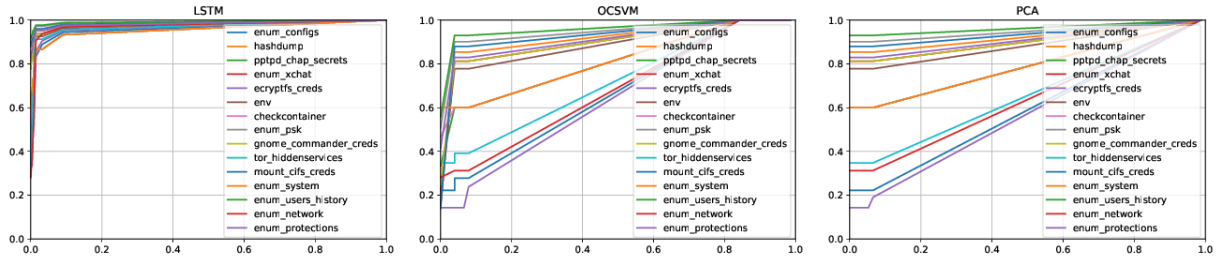



**Figure 8: ROC curves for Struts with CVE-2017-5638 for all three models.**

Several challenges arise in the practical deployment of AppMine. Privacy of user data is an important consideration that needs to be addressed for any system monitoring web applications. We believe that our aggregated feature representation (storing frequency vectors of system calls across time) provides some amount of protection against privacy leakage. Once more detailed information is extracted (such as system call parameters, and parameters from HTTP requests), the privacy risks to users increases significantly. It remains an open problem to quantify the exact amount of private information collected by a monitoring system, and to evaluate the tradeoffs between accuracy at attack detection and privacy implications on users.

In practical deployments, AppMine can be implemented as a security service offered by a cloud provider to its tenants running web applications in cloud-managed containers. We note that the cloud provider needs to train one ML model per web application

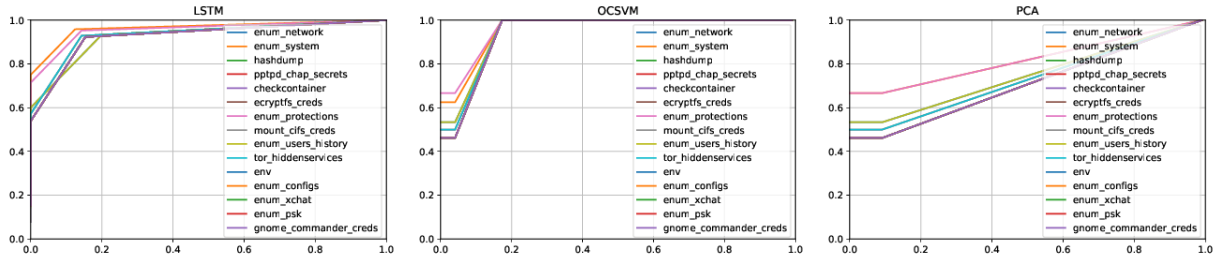



**Figure 9: ROC curves for Drupal for all three models.**

to detect anomalies that are specific to that web application. The training can be performed in parallel on distributed infrastructures for scalability.

Last, but not least, an attacker with knowledge of the AppMine design might attempt to perform an evasion or mimicry attack against the ML detector to avoid attack detection. That can be accomplished by preserving most of the legitimate application behavior and staggering the attack across longer time intervals. Understanding the susceptibility of LSTM-based models to evasion attacks in this context is an interesting topic, which we leave for future work.

:

## 6 RELATED WORK

Intrusion detection based on system call monitoring on end hosts has been studied in depth in the literature. Forrest et al. [15] create profiles of Unix process based on sequences of system calls and detect deviations under attack. Hofmeyr et al. [20] expand this by using fixed-length sequences and Hamming distances between unknown sequences and sequences in the normal database for measuring dissimilarities. Warrender et al. [50] introduce threshold-based sequence time delay embedding (t-STIDE), which implements an anomaly scoring technique where the score of a test sequence depends on the number of anomalous windows in the test sequence. Lane and Brodley [24, 25] use UNIX shell command sequences to build a user profile dictionary and propose various similarity measures to detect anomalous behavior. Lee et al. [26] propose an unsupervised learning method which generates association rules from training data (using RIPPER) and use these rules to detect anomalies in test data. Mutz et al. [33] use Bayesian earning methods to model system call parameters and detect anomalies in parameter values.

Ahmet et al. [4] propose anomaly detection models that use both spatial and temporal features extracted from Windows API calls. Markov models have been used to model the temporal sequence of system calls and detect anomalies (e.g., [29]). System call analysis has also been used for forensic investigation [36]. More recently, anomaly detection of system logs has been applied to enterprise networks [55] and cloud deployments [12]. Beehive [55] uses PCA and clustering-based methods to identify hosts with anomalous behavior in an enterprise networks, while DeepLog [12] designs an LSTM models that takes into account the sequence of system log events in HDFS and OpenStack logs to identify various anomalies.

In the area of web attacks, anomaly detection methods have been applied based on learning application profiles [23]. Bayesian networks were used to compose multiple models to reduce false positives [22]. Clustering of anomalies enables more detailed attack classification [40]. Spectogram [47] designs a mixture of Markov chains based on n-gram features to model the distribution of normal HTTP request parameters. Swaddler [9] models the worflow of an application and uses anomaly detection to identify when an application reaches an inconsistent state. Scarcity of training data for client web application was mitigated by Robertson et al. [39] by using global similarity of web requests. Several papers [30, 32, 45] use a hybrid approach based on program analysis and machine learning for web application vulnerability detection.

Recent work has shown how provenance, traditionally used in forensic investigation, has the potential to be used for intrusion detection. Han et al. [17] apply provenance for identifying anomalies in programs running in Platform-as-a-Service (PaaS) clouds. Winnower [18] provides an efficient mechanism for storing provenance data in a distributed cluster, and demonstrates attack detection capabilities in a Docker Swarm container cluster. CamFlow [34] and CamQuery [35] design new provenance architectures that speed up real-time applications, such as security monitoring. An interesting direction for future work is to investigate the use of these modern provenance architectures for web application vulnerability detection in cloud containers.

Other defenses against Web application vulnerabilities include the following: (1) static analysis (e.g., [21]); (2) dynamic analysis (e.g., [49]); (3) combination of static and dynamic analysis (e.g., [5]); (3) input validation (e.g., [43]); and (4) fuzz testing (e.g., [42]), but they have well-known limitations. For instance, most static analysis tools have large false positives, and input validation methods are specific to certain web attacks such as SQL injection. We believe that machine learning models can complement existing defenses in web applications deployed in either public or private clouds.

Industry solutions for container security in public and hybrid clouds include: Twistlock [3], a system for designing flexible access control policies for containers; Symantec Cloud Workload Protection [1], a system that provides better container isolation; and Trend Micro Deep Security [2], using ML to protect hybrid cloud workloads against vulnerabilities.

## 7 CONCLUSIONS

We propose an anomaly detection framework for detecting exploits in web application. We set up a testbed environment and deploy four web application, recreate seven exploits using Metasploit modules, and collect system call data using the Sysdig monitoring agent.

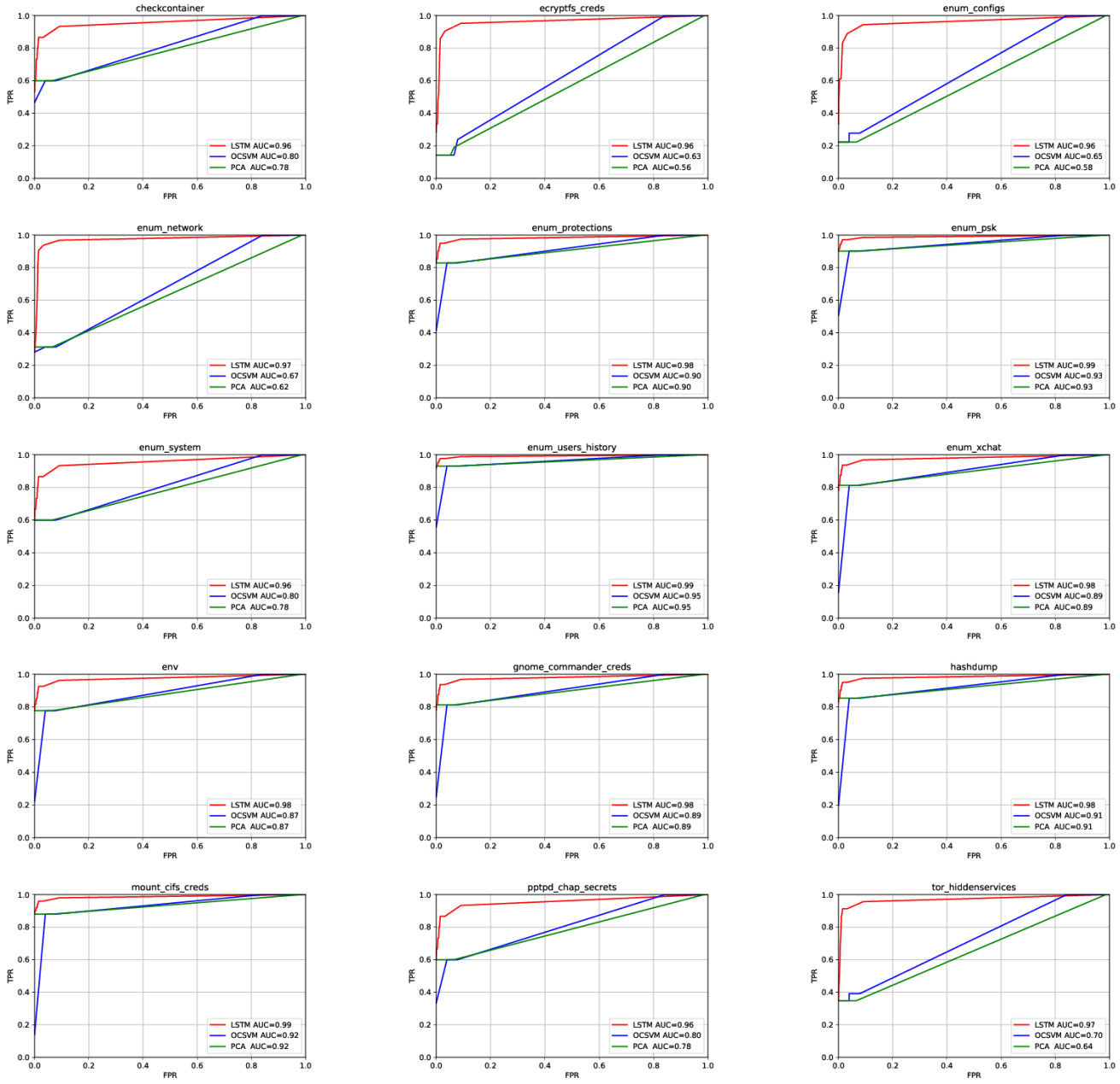



**Figure 10: ROC curves for attack scripts for Struts CVE-2017-5638**

We compare two traditional anomaly detection models (PCA and OCSVM) with an LSTM-model trained on sequences of system call frequency vectors. We demonstrate that LSTM outperforms the traditional models. Our framework has the advantage of not requiring attack data for training, and being applicable to a range of web application exploits. In future work, interesting research questions remain on extending this framework to other scenarios and running the algorithms in real cloud environments.

## ACKNOWLEDGEMENTS

We thank Vinny Parla, Andrew Zawadowskiy, and Donovan O'Hara from Cisco for suggesting the area of research and providing feedback and guidance during the course of the project. We also thank Haya Shulman for shepherding our paper and the anonymous reviewers for their constructive feedback. This project was funded by a research gift from Cisco, as well as the NSF grant CNS-1717634.

## REFERENCES


[1] Symantec Cloud Workload Protection. https://www.symantec.com/products/cloud-workload-protection.

[2] Trend Micro Deep Security. https://www.trendmicro.com/en_us/business/products/hybrid-cloud/deep-security.html.

[3] Twistlock. https://www.twistlock.com/.

[4] F. Ahmed, H. Hameed, M. Z. Shafiq, and M. Farooq. Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In *Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence*, AISec '09, pages 55–62, New York, NY, USA, 2009. ACM.

[5] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[6] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. DISCLOSURE: Detecting botnet Command-and-Control servers through large-scale NetFlow analysis. In *Proc. 28th Annual Computer Security Applications Conference (ACSAC)*, ACSAC, 2012.

[7] L. Bilge, E. Kirda, K. Christopher, and M. Balduzzi. EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proc. 18th Symposium on Network and Distributed System Security*, NDSS, 2011.

[8] C. M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[9] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna. Swaddler: An approach for the anomaly-based detection of state violations in web applications. In C. Kruegel, R. Lippmann, and A. Clark, editors, *Recent Advances in Intrusion Detection*, pages 63–86, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[10] Diamanti. 2018 container adoption benchmark survey. https://diamanti.com/wp-content/uploads/2018/07/WP_Diamanti_End-User_Survey_072818.pdf, 2018.

[11] Drupal. https://www.drupal.org/.

[12] M. Du, F. Li, G. Zheng, and V. Srikumar. DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1285–1298, New York, NY, USA, 2017. ACM.

[13] M. Dymshits, B. Myara, and D. Tolpin. Process monitoring on sequences of system call count vectors. In *Security Technology (ICCST), 2017 International Carnahan Conference on*, pages 1–5. IEEE, 2017.

[14] Cybersecurity incident and important consumer information. https://www.equifaxsecurity2017.com/, 2017.

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

[16] ftpbench. https://github.com/selectel/ftpbench.

[17] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer. FRAPpuccino: Fault-detection through runtime analysis of provenance. In *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*, Santa Clara, CA, July 2017. USENIX Association.

[18] W. U. Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *NDSS*, 2018.

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735—-1780, 1997.

[20] S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[21] N. Jovanovic, C. Kruegel, and E. Kirda. Pixy: A static analysis tool for detecting web application vulnerabilities (short paper). In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 258–263, Washington, DC, USA, 2006. IEEE Computer Society.

[22] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. Bayesian event classification for intrusion detection. In *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '03, 2003.

[23] C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Comput. Netw.*, 48(5):717–738, Aug. 2005.

[24] T. Lane and C. E. Brodley. An application of machine learning to anomaly detection. In *Proceedings of the 20th National Information Systems Security Conference*, volume 377, pages 366–380. Baltimore, USA, 1997.

[25] T. Lane, C. E. Brodley, et al. Sequence matching and learning in anomaly detection for computer security. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, pages 43–49. Providence, Rhode Island, 1997.

[26] W. Lee, S. J. Stolfo, et al. Data mining approaches for intrusion detection. In *USENIX Security Symposium*, pages 79–93. San Antonio, TX, 1998.

[27] L. Lenart. S2-052. https://cwiki.apache.org/confluence/display/WW/S2-052, 2017.

[28] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious URLs. In *Proc. 15th ACM International Conference on Knowledge Discovery and Data Mining*, KDD, 2009.

[29] F. Maggi, M. Matteucci, and S. Zanero. Detecting intrusions through system call sequence and argument analysis. *IEEE Trans. Dependable Secur. Comput.*, 7(4):381–395, Oct. 2010.

[30] I. Medeiros, N. Neves, and M. Correia. Detecting and removing web application vulnerabilities with static analysis and data mining. *IEEE Transactions on Reliability*, 65(1):54–69, March 2016.

[31] Metasploit. https://www.metasploit.com/.

[32] D. Mutz, W. Robertson, G. Vigna, and R. Kemmerer. Exploiting execution context for the detection of anomalous system calls. In C. Kruegel, R. Lippmann, and A. Clark, editors, *Recent Advances in Intrusion Detection*, pages 1–20, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[33] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel. Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):61–93, 2006.

[34] T. Pasquier, X. Han, M. Goldstein, T. Moyer, D. Eyers, M. Seltzer, and J. Bacon. Practical whole-system provenance capture. In *Proceedings of the 2017 Symposium on Cloud Computing*, SoCC '17, pages 405–418, New York, NY, USA, 2017. ACM.

[35] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer. Runtime analysis of whole-system provenance. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, pages 1601–1616, New York, NY, USA, 2018. ACM.

[36] S. Peisert, M. Bishop, S. Karin, and K. Marzullo. Analysis of computer intrusions using sequences of function calls. *Dependable and Secure Computing, IEEE Transactions on*, 4(2):137 –150, april-june 2007.

[37] The ProFTPD Project. http://www.proftpd.org/.

[38] RightScale. RightScale 2018 state of the cloud report. https://assets.rightscale.com/uploads/pdfs/RightScale-2018-State-of-the-Cloud-Report.pdf, 2018.

[39] W. Robertson, C. Kruegel, and G. Vigna. Effective anomaly detection with scarce training data. In *Proc. Network and Distributed System Security Symp. (NDSS)*, 2010.

[40] W. Robertson, G. Vigna, C. Kruegel, and R. A. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Proc. Network and Distributed System Security Symp. (NDSS)*, 2010.

[41] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323:533—-536, 1986.

[42] A. S., K. E., and K. C. Leveraging user interactions for in-depth testing of web applications. In *Proceedings of Recent Advances in Intrusion Detection*, RAID, 2008.

[43] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda. Preventing input validation vulnerabilities in web applications through automated type analysis. In *Proceedings of the 2012 IEEE 36th Annual Computer Software and Applications Conference*, COMPSAC '12, pages 233–243, Washington, DC, USA, 2012. IEEE Computer Society.

[44] Selenium - Web Browser Automation. https://www.seleniumhq.org/.

[45] L. K. Shar, L. C. Briand, and H. B. K. Tan. Web application vulnerability prediction using hybrid program analysis and machine learning. *IEEE Transactions on Dependable and Secure Computing*, 12(6):688–707, Nov 2015.

[46] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[47] Y. Song, A. D. Keromytis, and S. Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *Proc. Network and Distributed System Security Symp. (NDSS)*, 2009.

[48] Sysdig: Open Source Container Troubleshooting & Forensics. https://sysdig.com/opensource/.

[49] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-site scripting prevention with dynamic data tainting and static analysis. In *Proceedings of the 14th Annual Network & Distributed System Security Symposium*, NDSS, 2007.

[50] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344)*, pages 133–145. IEEE, 1999.

[51] Blog Tool, Publishing Platform, and CMS - WordPress. https://wordpress.org/.

[52] wpajaxloadmore. https://www.exploit-db.com/exploits/38660.

[53] wpnmedia. https://www.exploit-db.com/exploits/36810.

[54] wpreflexgallery. https://www.exploit-db.com/exploits/36809.

[55] T. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. K. Robertson, A. Juels, and E. Kirda. Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks. In *Annual Computer Security Applications Conference, ACSAC '13, New Orleans, LA, USA, December 9-13, 2013*, pages 199–208, 2013.


## A ROC CURVES FOR ATTACKS

In Figure 10 we show the ROC curves for PCA, OCSVM, and LSTM for all 15 attack scripts for the Struts application with the Equifax exploit (CVE-2017-5638).