Fast, Lightweight IoT Anomaly Detection Using Feature Pruning and PCA

John Carter
Department of Computer Science,
Drexel University, Philadelphia, PA
jmc683@drexel.edu

Spiros Mancoridis
Department of Computer Science,
Drexel University, Philadelphia, PA
mancors@drexel.edu

Erick Galinkin
Department of Computer Science,
Drexel University, Philadelphia, PA
eg657@drexel.edu

ABSTRACT

Anomaly detection is a method for identifying malware and other anomalies such as memory leaks on computing hosts and, more recently, Internet of Things (IoT) devices. Due to its lightweight resource use and efficacy, anomaly detection is a promising method to detect malware on small, resource-constrained hosts. Using Principal Component Analysis (PCA) to reduce the features, and hence the dimensionality of the anomaly detector, is common during the feature engineering process of classic machine learning methods, such as Support Vector Machines (SVM). However, as Neural Networks (NN) became more popular, many presumed that using PCA prior to using the data to train and deploy the model was unnecessary. In this work, we show that there is a significant advantage to using PCA for both SVM and NN-based anomaly detection. Doing so improves the performance and efficacy of malware detection models, and reduces the amount of data that needs to be stored on the device for on-device anomaly detection, thus making it useful for resource-constrained IoT devices. We also show that while pruning low-variance features may be an intuitive way to simplify a model, it is less effective than PCA to improve model training and deployment performance as well as model efficacy to detect malware.

CCS CONCEPTS

• **Security and privacy** → **Systems security**; **Malware and its mitigation**; **Intrusion detection systems**; **Embedded systems security**; • **Computing methodologies** → **Feature selection**;

KEYWORDS

Principal Component Analysis, Support Vector Machines, Neural Networks, Internet of Things, Malware detection, Anomaly Detection

ACM Reference Format:
John Carter, Spiros Mancoridis, and Erick Galinkin. 2022. Fast, Lightweight IoT Anomaly Detection Using Feature Pruning and PCA. In *The 37th ACM/SIGAPP Symposium on Applied Computing (SAC '22), April 25–29, 2022, Virtual Event.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3477314.3508377

1 INTRODUCTION

IoT devices are designed to execute a small number of specialized actions on lightweight computing devices. As a result, their behavior is limited and hence easy to learn. Moreover, because they are designed with only a few tasks in mind, their computational and memory resources are limited to the needs of those tasks. Behavioral anomaly detection is a popular and effective malware detection method, especially on resource-constrained devices such as IoT devices, due to its high level of efficacy and minimal resource consumption.

In this work, we have collected kernel-level system calls and network traffic from an IoT device on a local area network. Specifically, we collected data in cases where there is no malware running on the device as well as after two different malware samples had infected the device. We leverage the system calls and network traffic collected in these three different contexts as datasets for conducting anomaly detection on the device. These types of data provide a nearly comprehensive representation of the activities happening on the device, which means that each of them provide useful, explanatory features for behavior anomaly detection. We have also compiled a third dataset, which we call *combined features*, that merges both system call and network traffic data. It has been demonstrated [7] that such a combined dataset yields better detection performance than either of the other two datasets in isolation. We chose to collect behavioral data on a real device due to the lack of availability of comparable pre-existing datasets.

Since the merged data can grow quite large, but our device is highly storage-constrained, we seek ways to reduce the footprint of our security data. A meaningful way to address this is by performing feature engineering, which reduces the number of features that must be used to those with the best predictive capability. To this end, we consider Principal Component Analysis (PCA). PCA [8] is an algorithm that reduces the dimensionality of datasets by creating new uncorrelated variables from the original dataset attributes that maximize variance.

Many machine learning methods can be used for behavioral anomaly detection, and two popular methods are Support Vector Machines (SVM) and Neural Networks (NN). We demonstrate the results of using both a One-Class SVM and a NN on the data collected from an IoT device. One-class SVM is an unsupervised machine leaning method that learns a decision function that can classify new data as similar to or different from the training set data.

Our aim is to use this research to determine the feasibility of using feature compression to improve anomaly detection on IoT devices. To this end, we compare and explain the performance and efficacy of these algorithms when PCA is used for feature compression versus when it is not. We also compare the efficacy and performance of the algorithms when low variance attributes are pruned when PCA is used, or not used. This work includes the training and deployment cost trade-offs of using the aforementioned combination of options.

2 BACKGROUND

Detecting malware was once possible by solely comparing long bit-patterns extracted from the executable files of known malware and searching for these patterns in the executable files and process memory of computing hosts. However, with the advent of polymorphic malware, which encrypt most of the body of their executable files, and metamorphic malware, which mutate their executable files with every replication, classical anti-malware techniques became easier to defeat.

As a counter-measure to this evolution of malware, *behavioral malware detection* has become a common technique, where signatures are no longer restricted to bit-patterns found in the executable files of malware, but rather patterns of the malware execution behavior. Behavioral patterns can signify what malware are doing while executing, for example, reading from or writing to files, computing hashes, manipulating memory, starting or stopping system processes, or sending and receiving data from the network.

It is well-understood that sequences of system calls and network traffic traces provide insightful feature sets for behavioral anomaly detectors in networked computer systems. Since the system calls log the actions of a program on an operating system-level, they are useful to understand what is happening on a device in real time.

A bag-of-n-grams approach can be applied to the system call sequences, which breaks the system call sequences up and makes algorithmic learning more manageable [9]. Canzanese *et al.* [2] found success transforming raw system call data using Linear Discriminant Analysis (LDA) for dimensionality reduction for the problem of malware classification. Our work is concerned with anomaly detection, in contrast with Canzanese *et al*'s classification of labeled data. Consequently, LDA is not suitable for our task – as class separability is not critical for us, and so we use PCA instead.

Another way to detect anomalous behavior is to analyze network traffic data packets, which is the basis of network intrusion detection systems (NIDS) [1, 3, 4]. These systems look for specific behavior that is not common to the device, for example: rapid or high-volume packet transmission to or from another device; an unknown IP address communicating with the device; or a previously unseen protocol being sent from or to the device. The work of Hasan *et al.* [5] considers a variety of machine learning algorithms in the context of IoT anomaly detection with a focus on local network communications. Our work has similar aims, but addresses the on-device deployment model for anomaly detection with the intent of identifying malware infections on IoT hosts, while the research of Hasan *et al.* aims to create a NIDS model that can be deployed as a network sensor to identify network traffic anomalies.

3 EXPERIMENTAL SETUP

3.1 Dataset Acquisition

The data consist of system call traces and network traffic packets sent and received on an IoT router that is part of an IoT ecosystem, as shown in Figure 1. Two sensors are run during the data collection process: one to record system calls issued, and one to record network packets sent from or received by the device. System calls are the basic language of the device, in that it they are how processes can ask for resources from the OS kernel, and thus they provide excellent context for the events happening on the device. They can be found and logged in Linux using functions such as `strace`, and piped to a file to be analyzed later.

Similarly, network packets provide valuable context about any communication a device is having with another device. Each of these are useful indicators as to whether malware infection has occurred. These are recorded on-device by a package called CICFlowMeter [6]. CICFlowMeter is a package in the Python Package Index (PyPi) that listens to network traffic on a device, generates bidirectional network flows, and then extracts features from these flows.

Both of the sensors are run during (1) a period of benign behavior; (2) a period of ransomware running on the device; and (3) a period of a cryptominer running on the device, which together comprise the total dataset.

The ransomware malware traverses the device's directory tree starting at a user-specified directory and continually exfiltrates and encrypts each directory. The directories on the infected device are then unusable until they are decrypted using a key. Though ransomware does not often target IoT devices, there are scenarios in which it would be useful, such as if there are video files from an IoT camera or audio files from an Amazon Alexa that are sensitive in nature and stored on the device.

The cryptominer malware is tasked with continuously trying to mine coins by calculating hashes, and in our system, the cryptominer exports the computation results over the network. Though a single IoT device does not provide much computing power, many IoT devices can be linked together to collectively provide a lot of computing power for free, and thus can be a target for such malware.

These two malware families are meant to be examples of malware that could infect an IoT device, and are not meant to be a comprehensive set of malware that targets IoT devices. Both malware families were chosen specifically because of their frequent usage in the wild as well as the breadth of their behavior profiles. The ransomware represents malware that seek to gain information from the victim, while the cryptominer represents malware that seek to utilize the victim's computing power to complete some task. Although only two malware samples were used in this work, each malware sample represents a family of malware that is parameterized by its data exfiltration rate.

We chose to use this data due to the lack of dataset availability for both benign and malicious behaviors on an IoT device. The benign data consists of communication between two other components of the ecosystem: an IoT camera and a client app viewing the camera's live video feed. The malware data consists of system calls and network traffic generated by the malware's execution in addition to the normal, benign use of the IoT camera and client app.

Although the router was chosen for collecting the datasets used in this work, the system call traces and network traffic packets could have been collected on any of the devices in the ecosystem, and the process would be the same modulo the installation of the additional functionality. One of the benefits of this interchangeability is that the anomaly detection process can be modified to suit the particular device on which it is running. For example, the amount of collected data stored on the device could be increased or decreased given the storage constraints imposed by a particular device's hardware.

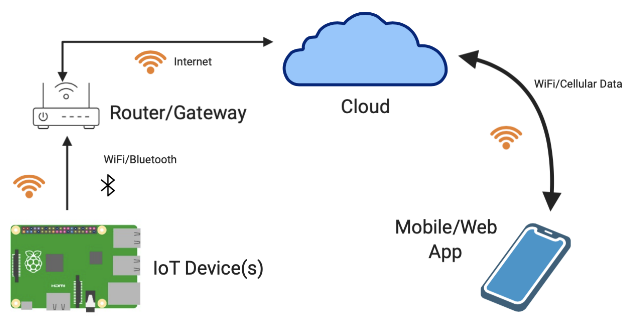

Figure 1: The IoT Ecosystem

3.2 Machine Learning Models

Two models were used in this research: a One-Class Support Vector Machine, and a shallow Neural Network. These were chosen because they are both are lightweight in terms of their resource consumption and the amount of data each requires for high-efficacy anomaly detection, which make them useful for small, lightweight devices like IoT devices.

The SVM used is scikit-learn's One-Class SVM, and it uses the default Radial Basis Function (RBF) kernel. The SVM uses two hyperparameters: $nu = 0.1$ and $gamma = 0.1$.

The NN is built using PyTorch, and includes three layers: the input and output layers, and one hidden layer. A ReLU activation function is applied after the input layer and the hidden layer, and after that Batch Normalization is applied to each. Dropout is performed prior to the output layer, which provides a binary classification via a Sigmoid activation function. Since the problem is a binary prediction, the network leverages a binary crossentropy loss. The Adam optimizer is used in the model. The learning rate for the NN is 0.001, the batch size is 64, and the number of training epochs is 100.

4 DATA PROCESSING AND FEATURE ENGINEERING

Feature engineering is an important step in the machine learning pipeline since it extracts the most explanatory and useful features from the raw data collected. The more explanatory the features are, the better the machine learning models perform at their task – in this case, detecting anomalies possibly due to malware infection.

One aspect of this process is removing features that are not explanatory, which makes the model training and deployment more efficient. This is crucial for deploying the models on resource-constrained IoT devices, which are typically designed for single, specialized applications.

The processing times in subsequent sections reflect an offline training and deployment process for the model, where the data is offloaded from the device and processed, used to train the model, and the trained model is deployed to the device. This was done for speed and ease of iteration and experimentation.

However, we have verified that the processing can also run entirely on-device, albeit at an increased processing time. The raw data files that contain system call logs and packet transmission information can be large, such as the raw data files listed in Table 1. The file sizes indicate the size of the data collected over a period of 7 minutes.

	File Size
Benign Syscall	2602KB
Ransomware Syscall	6954KB
Cryptominer Syscall	23976KB
Benign Network	103KB
Ransomware Network	153KB
Cryptominer Network	222KB

Table 1: Initial raw data file sizes

4.1 Initial Data Processing

We begin by detailing the initial data processing step, which involves grouping the collected data into segments that are more useful for anomaly detection.

System call traces and network traffic were collected during each time period. After the raw data was collected, the system calls were grouped by timestamp using a user-specified window size. In this case, the window size was 5 seconds, which means the 7 minutes of data collection is split into 5 second intervals, and both system calls and network traffic are included in each interval.

A bag-of-n-grams approach was then used on the groupings, where the value of n was also user-specified. This means that the feature set is composed of the number of observations of each n consecutive system calls in a particular time window. Through empirical analysis, we found that a value of $n = 2$ is optimal for both performance efficiency and detection efficacy on our dataset. A value of $n = 1$ is often less useful for this type of analysis because it does not provide enough context to establish a pattern between successive uses of the same or different system calls. On the other hand, through experimentation, we have found that using a value of $n \geq 3$ does not yield significantly better results than using bi-grams.

The data grouping is repeated for the network traffic as well, where each of the features are grouped into m-second intervals, which may differ from the interval used for syscalls. The goal is to make the packet sequences more distinct by finding the optimal value of m such that the interval is large enough to collect useful data on a particular stream of packets, *e.g.,* source/destination IP, but small enough that other streams of packets are excluded. In this experiment, we choose $m = 5$ seconds for the network packets.

4.2 Feature Pruning

Intuitively, dataset features with low variance are less likely to be predictive and can be pruned from the dataset with little or no impact on anomaly detection efficacy. Ideally, this reduces the storage space required by processed data and creates better explanatory datasets for anomaly detection models.

To prune the features, we calculated the variance for each feature, and used a variance threshold of 0.5 to determine if a particular feature has enough variance to be left in the feature set. In some datasets, a majority of the features fell below this threshold and were pruned, as depicted in Figure 2. We arrived at a threshold of 0.5 through experimentation that retained no less than 20% of the dataset and no more than 80% of the dataset. This ensures we drop some number of features to improve storage but not so many that the model will be overfit to a very small number of higher-variance features.

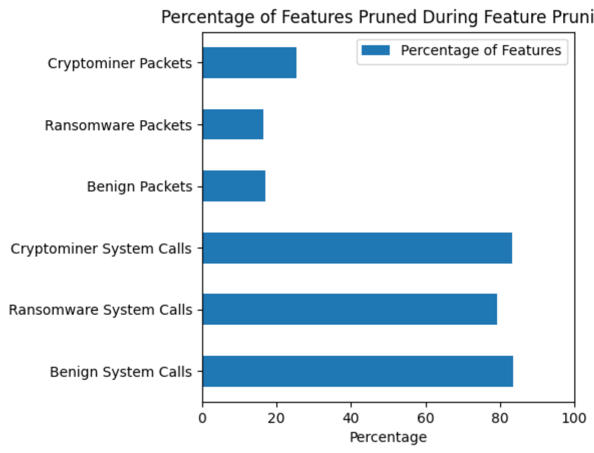

Figure 2: Percentage of features pruned

4.3 Principal Component Analysis

To generate useful features from the full dataset, we employed PCA in two different test cases. One dataset leveraging PCA was generated after the initial data processing, and a second dataset leveraging PCA was generated after the feature pruning process. This yielded a total of four datasets, as seen in Table 2.

We seek to demonstrate the effect that both feature pruning and PCA have on the processed data, in terms of model performance as well as model efficacy. As shown in Table 2, using PCA on the raw data after completing the initial data processing dramatically reduces the storage needed. In this example, the processed data that used PCA required approximately 1% of the storage used by processed data that did not use PCA.

Figure 3 shows the performance benefit gained in both model training and model deployment.

Finally, Figure 4 shows the benefits PCA provides in terms of increased detection rates, especially for the NN model.

5 MODEL TRAINING AND DEPLOYMENT EVALUATION

The model training and model deployment process for each dataset was conducted 10 times, and the average training/deployment costs and detection rates for those 10 trials are shown in Table 3 and Figure 4.

The test data for the models are balanced in terms of the number of benign observations versus the number of malware observations, and the two types of observations are randomly interspersed. This is important to note, because if the dataset were imbalanced, it would likely be biased towards the benign dataset. As a result, it would not learn the difference between benign and malicious data as well, and likely cause the model to classify everything as benign since it found that to be the more common target class.

We observe from Table 3 that running PCA on the full dataset prior to training a NN benefits performance by almost an order of magnitude. This provides considerable improvement above and beyond pruning low variance features. Figure 3 further illustrates this point by showing that using PCA on NN-based anomaly detectors greatly reduces both the training and deployment costs. Although the ability to train models on-device was verified, the costs here reflect an off-device training process.

Table 3 shows that running PCA on the dataset increases the training cost of the SVM slightly. Albeit, the SVM has a superior malware detection efficacy when PCA is used. Additionally, the training and deployment cost of the SVM, as illustrated in Figure 3, is between 1 and 1.5ms, approximately 1000 times faster than the NN. Although the training and deployment cost of the SVM does not follow a downward trend between using all features to using reduced features in the same way the NN does, we believe the cost is so minimal that it is within a rounding error and thus not significant.

Figure 4 demonstrates that the SVM performance was consistently perfect across datasets, achieving 100% anomaly detection on nearly all datasets. This includes datasets that used all of the features (*i.e.,* neither feature pruning nor PCA usage) and datasets that used PCA to reduce the features, which resulted in the fastest trained and deployed anomaly detectors.

The neural network achieved less robust detection rates on anomaly detection in general, with a maximum average anomaly detection rate of 98%. It also had poor performance on the reduced packet features, after the low variance features were pruned (without using PCA), where it achieved an average anomaly detection rate of only 81%.

These results are in line with much of the research leveraging SVMs for lightweight anomaly detection, where one-class SVMs perform incredibly well.

We note that although all anomalies were detected by SVMs with the majority of these datasets, in uncontrolled environments, not all anomalies are necessarily malicious. This could lead to false positives in cases where anomalies trigger a response, the exploration of which we reserve for future work.

	All	**All+PCA**	**Dropped**	**Dropped+PCA**
	Time/Size	Time/Size	Time/Size	Time/Size
Benign Syscall	438/322	414/3	429/53	441/3
Ransomware Syscall	677/362	658/3	655/76	682/3
Cryptominer Syscall	2200/1432	2097/10	2189/244	2020/10
Benign Network	5919/89	6387/3	6072/77	6240/3
Ransomware Network	6602/138	6560/3	6266/125	6350/3
Cryptominer Network	6841/143	6513/3	6964/123	6726/3
Benign Combined	6566/345	6815/5	6660/119	6696/5
Ransomware Combined	7506/423	7234/5	7100/186	6994/5
Cryptominer Combined	9000/558	8627/6	9320/196	8762/6

Table 2: Performance statistics for processing various feature sets. Times are in ms, sizes are in KB.

	Syscalls	**Packets**	**Combined**
	(Ransomware/Cryptominer)		
	Support Vector Machine		
All features without PCA	1.5/0.9	1.2/1.0	1.1/0.8
Reduced features without PCA	2.3/1.0	1.8/1.0	1.1/0.9
All features with PCA	1.4/0.95	1.2/1.0	1.7/1.0
Reduced features with PCA	1.1/0.9	0.9/0.9	1.5/0.9
	Neural Network		
All features without PCA	910/1754	570/601	1170/1431
Reduced features without PCA	620/1123	620/611	690/692
All features with PCA	490/869	470/462	470/504
Reduced features with PCA	470/927	450/501	460/527

Table 3: Training cost of anomaly detection models. Times are in ms.

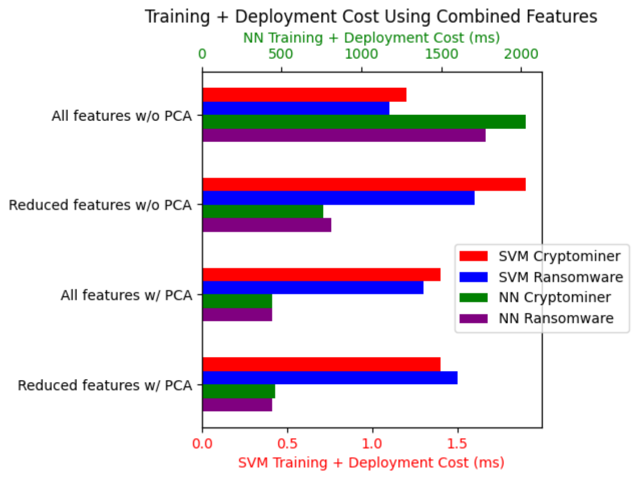

Figure 3: Training + Deployment Costs for SVM and NN using the combined feature set

In addition to the high anomaly detection rate of the SVM across the board, a significant finding is that the processed feature memory size for all features remains quite small, at only 10KB for the full training data set. This suggests that a baseline feature set could easily be captured and stored on-device and used to quickly train an anomaly detector. This anomaly detector could then capture further baselines and re-train according to some environment-specific schedule. Since the data processing can be done on-device, ahead of the model training, this allows for an IoT device to commit the majority of its resources to its intended task while also providing the capability to train and run a lightweight, on-device anomaly detector.

PCA improved the detection rates of the neural network on all datasets, as seen in Figure 4. We can also observe that the NN's detection rate was improved in the syscall and combined datasets when low variance features were pruned from the dataset.

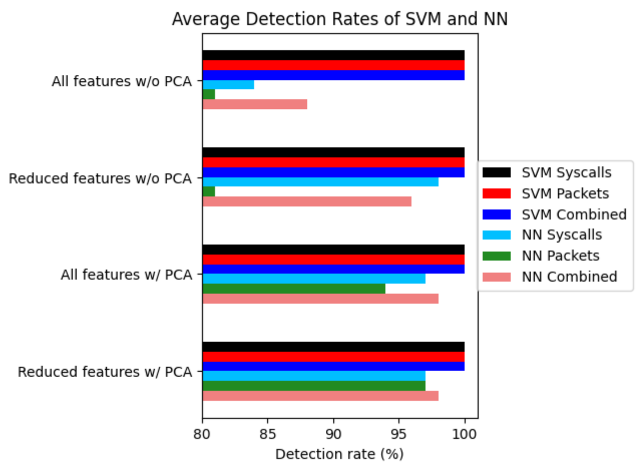

Figure 4: Average detection rate of the two malware families using SVM and NN.

From Table 2, we see that there is a significant reduction in the size of the syscall data when PCA, low variance feature pruning, or both are applied. This directly corresponds with improved detection rates, suggesting that, at least in the case of syscall data, we are capturing large amounts of noise that neural networks struggle to find the signal in.

This finding is important even outside of the realm of IoT and strongly suggests that despite the success of using raw data for tasks like natural language processing and computer vision, when using high-volume data with a lot of noise, it remains important to identify and extract useful features.

6 CONCLUSION

In this work, we collected system call data and network traffic data from a real IoT device, used two different methods for dimensionality reduction on the data, and evaluated the effects of both methods using two popular machine learning models.

While both PCA and low-variance feature pruning were successful in reducing the amount of storage needed, PCA was much better at reducing the dimensionality of the data such that training and deployment time was decreased and the efficacy of the models was increased.

While PCA was common during the feature engineering process of classic machine learning methods, such as SVMs, many presume that PCA is no longer needed for NNs. We demonstrate that this is not the case, and that using PCA in feature engineering prior to training a NN improves the run-time deployment performance and efficacy of the model.

We further demonstrate that despite the state of the art achievements of neural networks on a variety of tasks, simple models like one-class SVMs continue to provide a faster-training, more lightweight, more accurate anomaly detector.

We acknowledge the risk of introducing false positive detections with an anomaly detection rate of 100%, though no alerts were triggered on benign traffic run through the SVM. In future work, we aim to explore a data collection and model re-training schedule for environments where this system might be deployed.

In this highly controlled environment, the introduction of malware is a deliberate event. However, in most environments, the introduction of new devices and the introduction of malware is not well-controlled. This future work would consider how to control for model drift and deal with the potential risk of false positives, which are notoriously distracting for security teams. By identifying times or triggers for re-training the lightweight, on-device anomaly detectors, we can improve the detection of and response to anomalous events.

Meaningful future work can also be conducted on the selection of time window and n-gram size. In our case, syscall bi-grams and a window size of 5 seconds for both network traffic and syscalls proved highly effective. However, these results were discovered through experimentation and familiarity with the system, environment, and implementation. We will seek to explore ways that these optimal window sizes and selection of n in our n-grams can be determined automatically and ideally, dynamically. A training-time selection of window and n-gram size may allow the system to adapt to a wider variety of environments without needing a human to specify these values, which would make deploying the system on other types of devices even more seamless.

7 ACKNOWLEDGEMENT

This research was funded by the Auerbach Berger Chair of Cybersecurity held by Spiros Mancoridis.

REFERENCES

[1] Monowar H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita. 2014. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys Tutorials* 16, 1 (2014), 303–336. https://doi.org/10.1109/SURV.2013.052213.00046
[2] Raymond Canzanese, Spiros Mancoridis, and Moshe Kam. 2015. Run-time classification of malicious processes using system call analysis. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 21–28.
[3] Raghavendra Chalapathy and Sanjay Chawla. 2019. Deep Learning for Anomaly Detection: A Survey. *CoRR* abs/1901.03407 (2019). arXiv:1901.03407 http://arxiv.org/abs/1901.03407
[4] Pedro García-Teodoro, Jesús Díaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security* 28 (02 2009), 18–28. https://doi.org/10.1016/j.cose.2008.08.003
[5] Mahmudul Hasan, Md Milon Islam, Md Ishrak Islam Zarif, and MMA Hashem. 2019. Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches. *Internet of Things* 7 (2019), 100059.
[6] Hieu Le. 2021. CICFlowMeter. https://gitlab.com/hieulw/cicflowmeter.
[7] Mashid Noorani, Spiros Mancoridis, and Steven Weber. 2019. On the Detection of Malware on Virtual Assistants Based on Behavioral Anomalies.
[8] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2, 11 (1901), 559–572.
[9] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. 2001. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001.* 144–155. https://doi.org/10.1109/SECPRI.2001.924295