**Malware detection for IoT using One-Class Classification**

**1 Introduction**

In the burgeoning field of the Internet of Things (IoT), the increase of interconnected devices has dramatically changed the way people interact with modern technology. It offers marked improvements in efficiency and utility in daily life. But IoT is not limited to private consumption, its deployment in industrial settings is also redefining manufacturing, logistics, and energy distribution realms. The Industrial IoT (IIoT), typified by sensors and automation, is aimed at enhancing operational effectiveness, predictive maintenance, and live monitoring across diverse sectors. Despite these advancements, the escalation in IoT and IIoT deployments is matched by an upsurge in cybersecurity vulnerabilities, such as IoT malware. These threats bear significant implications, with the potential to disrupt not only consumer devices but also vital industrial operations. A security breach could have dire outcomes, ranging from personal data exposure to the crippling of fundamental infrastructure, even escalating to widespread, coordinated attacks on crucial systems.

IoT malware presents distinct challenges that stem from the heterogeneity and widespread presence of devices, from household gadgets to advanced industrial controls. The design of many IoT devices, especially those in industrial contexts, emphasizes utility over security, rendering them susceptible to assorted cyber threats. For example, the infamous IoT Botnet Malware Mirai, heavily relies on the default system password of these Linux based IoT devices for self-propagation. Thus, the detection of IoT malware needs to be addressed immediately. Robust, efficient detection methods are essential to ensure the integrity, confidentiality, and availability of IoT infrastructures.

In tackling such security concerns, the application of anomaly detection which is a type unsupervised learning stands out as a feasible technique. Contrary to conventional training on labeled datasets, unsupervised learning utilizes the vast, unlabeled network data generated by IoT systems in their normal operation. This strategy affords several benefits in the IoT sphere. Primarily, the immense and diverse data generated by innumerable devices are not labeled, labeling these data for supervised learning can be slow and resource intensive. Additionally, IoT's dynamic environment and evolving malware methodologies quickly outdated static, labeled datasets. In contrast, unsupervised learning can nimbly accommodate shifts and trends within real-time data streams, offering a more robust and adaptive solution for malware detection.

In this study, we examine the nuances of IoT malware data and the function of detection frameworks in protecting IoT devices. We evaluate the pros and cons for unsupervised learning techniques and illustrate their potential in meeting the complex requirements of IoT security in a world of growing connectivity. We aim to contribute to this study as follows:

* Focus on high recall: Our study focuses on achieving higher possible recall with an acceptable lower rate in precision. In some cases, such as critical industrial or military, catch as much malware as possible is crucial.
* Exploring on One Class Training: Our proposed methodology concentrates on training with only benign data, a form of one-class classification, to simulate real-world environments where malicious data are not always accessible. This approach also lowers the impact of a specific type of malware that could potentially be used in the training data.
* Prioritizing Deployment Compatibility: Our design strategy emphasizes compatibility and easy deployment on a wide array of IoT devices, irrespective of their computational limitations, to guarantee broad-based efficacy without compromising device performance.

**2 Related Work**

**2.1 Supervised learning**

IoT security has gained significant attention in recent years, with numerous studies focusing on malware detection using various machine learning techniques. However, traditional approaches have predominantly relied on supervised learning methods for high accuracy because it is much easier to access a public dataset which is often labeled. For studies that generate their own data, the size is usually small, so it won’t be expensive to label. Pajouh et al. [1] proposed a supervised machine learning model, showcasing the efficiency of supervised learning algorithms in detecting malware with high accuracy but also pointing towards challenges in handling evolving malware threats. Although they have achieved decent results, 96% detection accuracy(precision) and 4% false positive, they have used a relatively small dataset which contains 152 malware and 450 benign and then they used SMOTE to up sampling the data size to double, triple and quintuple. Sudheera et al. [2] used a very large dataset that contains multiple scenarios, their goal is to detect and identify the attack stages in IoT networks. They also used supervised learning algorithms such as SVM, KNN, and RF. They have achieved 99% accuracy in four datasets that are different mix from different scenarios in the large dataset.

Deep learning is also being used in previous studies, and different frameworks have been proposed to improve the performance of malware detection. Deep learning has multiple advantages over traditional machine learning that are deep features, continuous learning, and allow more complex and customized structures. Sahu et al. [3] proposed a hybrid model that uses CNN to extract high level features and then classify using LSTM and they have reached 96% accuracy (97% recall on malware) for all types of attacks in the dataset. There are also studies that use deep learning on images representations of malwares like Cui et al., [12] and Vinayakumar et al., [13]. The later study also addressed the issue of biased training by using existing malware samples in the training data. They proposed a novel image processing technique to mitigate this issue.

DeepAM is heterogeneous deep learning framework for intelligent malware detection by Yanfang Ye et al., [11] explores a deep learning architecture for malware detection using AutoEncoders and multilayer restricted Boltzmann machines, showing improvement over traditional methods. Although they have used autoencoder which can be used as an anomaly detection deep learning structure, they used it as a feature compression layer instead of use it as an unsupervised anomaly detection tool.

**2.2 Unsupervised learning**

In real world scenarios, labeled data would require extra effort and cost which are often not applicable. Fang et al. [4] discussed the vulnerability of models based on supervised learning to specific attacks, presenting a reinforcement learning framework to evade anti-malware engines, highlighting the limitations of supervised learning in adapting to sophisticated malware modifications. Jahromi et al. [5] demonstrate an enhanced LSTM method for malware threat hunting, leveraging supervised learning for accurate detection, yet emphasizing the need for robust methods against new malware variants. A knowledge transfer-based semi-supervised federated learning framework for IoT malware detection is explored by Pei et al. [6], indicating the potential of combining supervised and unsupervised learning to mitigate data labeling challenges. However, semi-supervised learning also requires a portion of labeled data.

In Pu et al. ’s [14] study, they have presented an unsupervised anomaly detection method that combines sub-space clustering and one class SVM to detect attacks without any prior knowledge and they stated that they have achieved better performance for existing techniques. Zhang et al. [15] proposed a novel unsupervised model based on adversarial autoencoder and deep clustering, they utilized deep clustering to preserve the loss of the dynamic encoder reconstruction and improved the results 42.2% based on traditional PCA feature extraction methods. However, these clustering methods are based on a pre-defined threshold that would still require using malware and benign samples in the training set and rely on the labels which are not used in training to define the threshold.

**2.3 One class classification**

Acquiring malware samples, especially real-world malware samples, could also be challenging. So, study of the feasibility of one class classification which only requires unlabeled benign samples could also be useful. Tajoddin et al. presented RAMD, a registry-based anomaly malware detection using one-class ensemble classifiers. Mahmoud Al-Qudah et al. [7] presents an enhanced classification model based on One class SVM (OCSVM) classifier for detecting deviations from normal memory dump file patterns as malware. It integrates OCSVM and PCA for increased sensitivity and efficiency.

**3 Data**

The dataset used in this project is called IoT-23[9]. This dataset is released in January 2020 and compiled by the Stratosphere Laboratory at CTU University in the Czech Republic and funded by Avast Software. This dataset comprises 23 different network traffic captures (scenarios), 20 are from IoT devices infected with malware, each labeled with the malware name, and 3 from benign IoT devices including a Philips HUE smart lamp, an Amazon Echo, and a Somfy smart door lock. These captures were made on real hardware devices in a controlled environment with normal internet access. The dataset includes detailed information such as README.md files, original pcap files, Zeek conn.log.labeled files with network flow and behavior labels, and other analysis files. The coon.log.labeled files which are the connection logs based on NetFlow analyzed by Zeek. Different than network traffic in the pcap files, NetFlow is a network protocol developed by Cisco for collecting IP traffic information and monitoring network flow. Cisco standard NetFlow version 5 defines a flow as a unidirectional sequence of packets that all share seven values which define a unique key for the flow. The seven attributes are: ingress interface (SNMP ifIndex), source IP address, destination IP address, IP protocol number, source port for UDP or TCP (0 for other protocols), destination port for UDP or TCP (type and code for ICMP, or 0 for other protocols), and IP Type of Service.

A table of information

Description automatically generated

Table 1.

There are originally 20 features in the labeled dataset as Table 1 shows. Table 2 displays the number of samples for each type of attack including benign samples across all 23 scenarios. The type of attack is classified in the original data. Because a large portion of the types contain very little number of samples, some combinations of labels are being performed. Okiru-Attack is combined into Okiru, C&C-PartOfAHorizontalPortScan is combined into PartOfAHorizontalPortScan, and all the reset types except the ones mentioned, Benign, and DDoS, are combined into Attack.

|  |  |
| --- | --- |
| Type | Amount |
| PartOfAHorizontalPortScan | 156,528,412 |
| Benign | 33,994,932 |
| Okiru | 33,750,546 |
| DDoS | 17,097,944 |
| Okiru-Attack | 13,609,470 |
| C&C | 20,058 |
| C&C-HeartBeat | 21,605 |
| Attack | 6,643 |
| C&C-FileDownload | 36 |
| C&C-HeartBeat-Attack | 834 |
| C&C-HeartBeat-FileDownload | 11 |
| C&C-PartOfAHorizontalPortScan | 888 |
| C&C-Torii | 30 |
| FileDownload | 4 |
| PartOfAHorizontalPortScan-Attack | 5 |

Table 2.

**4 Methodology**

**4.1 Data processing and feature engineering**

The data from both captures are merged and processed together to ensure consistency across all features. Distinct labels are assigned to samples from different captures to facilitate easy separation later. The conn.log.labeled file is treated as a CSV file containing 21 features, with 20 features listed in Table 1 and the last feature being the detailed label. In this project, the original detailed label, which specifies the type of malware attack for each sample, is simplified to a binary label indicating benign or malicious. Consequently, the detailed label is not used. Features such as the timestamp and UID, which do not contribute to the analysis, are removed. Additionally, features local\_orig and local\_resp, which are entirely empty, and the services feature, which contains over 99% missing values, are also removed. This leaves 15 features, comprising 7 numeric and 8 categorical features. The categorical features are encoded into numeric values to facilitate processing by the algorithm.

The remaining categorical features, excluding connection state and connection history, are IP addresses and port numbers. Port numbers are categorized into three labels: well-known ports (0-1023), registered ports (1024-49151), and dynamic/private ports (49152-65535). For IP addresses, given the extremely low appearance of IPv6 addresses in the datasets used, each address is split into four separate numeric features, and IPv6 addresses are split into 4 zeros. This approach is supported by Shao et al.’s study [10], which found that splitting IPv4 addresses into four numeric features performs better than one-hot encoding. Unlike their study, which dealt with a limited number of different IP addresses, the dataset in this study contains a large variety of IP addresses, making one-hot encoding impractical.

|  |  |
| --- | --- |
| conn\_state | Summarized state |
| S0 | Connection attempt seen, no reply |
| S1 | Connection established, not terminated (0 byte counts) |
| SF | Normal establish & termination (>0 byte counts) |
| REJ | Connection attempt rejected |
| S2 | Established, Orig attempts close, no reply from Resp |
| S3 | Established, Resp attempts close, no reply from Orig |
| RSTO | Established, Orig aborted (RST) |
| RSTR | Established, Resp aborted (RST) |
| RSTOS0 | Orig sent SYN then RST; no Resp SYN-ACK |
| RSTRH | Orig sent SYN-ACK then RST; no Orig SYN |
| SH | Orig sent SYN then FIN; no Resp SYN-ACK ("half-open") |
| SHR | Resp sent SYN-ACK then FIN; no Orig SYN |
| OTH | No SYN, not closed. Midstream traffic. Partial connection. |

Table 3-1.

|  |  |
| --- | --- |
| history | Description |
| S | A SYN without the ACK bit set |
| H | A SYN-ACK ("handshake") |
| A | A pure ACK |
| D | Packet with payload ("data") |
| F | Packet with FIN bit set |
| R | Packet with RST bit set |
| C | Packet with a bad checksum |
| I | Inconsistent packet (Both SYN & RST) |
| Q | Multi-flag packet (SYN & FIN or SYN + RST) |
| T | Retransmitted packet |
| W | Packet with zero window advertisement |
| A | Flipped connection |

Table 3-2.

Handling the features conn\_state and history can be challenging, requiring a deeper understanding of networking and cybersecurity. For conn\_state, direct one-hot encoding is feasible due to the limited number of labels. However, for history, the values can be random strings composed of uppercase and lowercase letters, as shown in Table 3. Although conn\_state has a limited number of labels, adding more than 10 dimensions to the dataset is not ideal, especially when some labels have very few occurrences compared to the most frequent label Therefore, for conn\_state, labels containing RST are combined into a new label RST, as they all represent an unusual reset request for the TCP connection. Similarly, labels S0 and SF are combined into one label, as they essentially represent the same condition with a minor difference in byte transformation.

A more in-depth analysis is needed for connection history. A random string pattern that consists of the same set of letters as shown in Table 3. can be processed by term frequency and inverse document frequency (tf-idf) which are often used in nature language processing that aims to measure the importance of a word to a document in a collection of corpuses. In this case, the word is each letter, and the corpuses are the random string patterns. After apply this technique to the history column, it will result in 12 new features which are numerical value for each letter.

At this stage, there are 6 labels each in conn\_state and history, and 3 labels in proto, allowing for one-hot encoding to be applied. Upon further examination of the numeric features, it is found that some features contain more than 99% of the same value which indicates that these features are very likely to have low impact on the target. To ensure extreme cases that these types of features could also be characteristic, the frequency is calculated separately for each type of label, and if a feature contains over 99% of the same value for all types including benign, it will be removed. Consequently, the final dataset comprises 18 features.

**4.2 Anomaly detection techniques**

Most previous studies have traditionally relied on supervised learning methods, necessitating well-labeled datasets for training to build hybrid models for static or dynamic analysis. This conventional approach, while beneficial in structured environments, encounters significant challenges in the IoT context, where labeling vast amounts of data is impractical due to their sheer volume and heterogeneity. Only a limited number of studies have ventured into exploring the use of unlabeled data which is project aims to address because there remains a significant research void in comprehensively employing these unsupervised methods for IoT malware detection that becomes more important given the practical difficulties in obtaining labeled data in real-world settings.

Detecting malware is essentially an anomaly detection problem, however most previous studies have been using both malicious and benign data for training which is logical because the results will be more accurate. However, in real world situations, malicious data are not always accessible especially for a real time malware detection system, malicious data are most likely not available, the training must be done only based on the normal benign data. This project delves into an underexplored territory in IoT cybersecurity research — the exclusive use of unlabeled positive data for training detection models and aim to build on the limited existing research and advance the field by proposing methodologies that are specifically tailored to exploit unlabeled data, ensuring both the practicality and effectiveness of malware detection in diverse IoT settings. This focus on unlabeled data represents a significant departure from traditional methods and positions our study as a crucial contribution to the evolving landscape of IoT cybersecurity.

One-class SVM and isolation forest are the focus in this section, both algorithms are proven to be effective for anomaly detection problems. One-Class SVM is adapted from the traditional SVM, specifically designed for anomaly detection. It works by finding a decision boundary that separates most of the data points from the outliers. Isolation Forest is another unsupervised learning method for anomaly detection. It isolates anomalies instead of profiling normal data points. It works by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. In most cases, one-class SVM has been proven to have more accuracy than isolation forest, however, in this case where the dimensions are high, SVM could encounter significant performance decrease, especially for the non-linear kernels. Isolation forest on the other hand has negligible performance impact with higher dimensions.

Auto encoder is a structure of deep learning anomaly detection. It typically consists of two main parts, an encoder, and a decoder. The encoder compresses the input into a latent-space representation, and the decoder reconstructs the input data from this representation. The aim is to learn a representation that captures the most important features of the data and detect anomalies by the reconstruct error.

A diagram of a computer code

Description automatically generated

Image 1.

In this study, an autoencoder with a sequential architecture was employed for malware detection through anomaly detection. The model comprises six densely connected layers. The input layer has 17 neurons, corresponding to the dimensionality of the input data. The encoder part of the autoencoder consists of two hidden layers with 16 and 8 neurons, respectively, reducing the dimensionality of the input data to a compressed representation. The bottleneck layer, the smallest layer in the network, has 4 neurons. The decoder part mirrors the encoder, with two hidden layers of 8 and 16 neurons, respectively, aiming to reconstruct the input data from the compressed representation. The output layer has 17 neurons, matching the number of input features, and is used to compare the reconstructed data with the original input. The model has a total of 933 trainable parameters, making it a relatively lightweight neural network suitable for detecting anomalies in the context of malware detection.

In the context of determine the reconstruction error threshold to classify anomalies, using the Median Absolute Deviation (MAD) instead of percentiles to determine the threshold is a more robust approach. MAD is a measure of variability that is less sensitive to outliers in the data, making it a suitable choice for anomaly detection tasks where the goal is to identify rare or unusual observations. By calculating the MAD of the reconstruction errors between the original input and the reconstructed output of the autoencoder, a threshold can be set based on a multiple of the MAD value. This threshold can then be used to classify observations as normal or anomalous. Using MAD as the basis for the threshold helps to reduce the impact of extreme values in the data, leading to more stable and reliable anomaly detection. This approach is more robust and scalable than the percentiles method and can potentially achieve better performance for unseen data.

Let X be the dataset, the first step is to get the median for the data set, then D represents the array of absolute deviations from the median, and lastly, the MAD score is equal to the median of D. In a normal distribution, the MAD is related to the standard deviation (σ) by the equation.

And we get the MAD-score which is a z-score using the following equation.

**4.3 One-class classification**

One-class classification and anomaly detection are closely related concepts in machine learning, often used interchangeably, yet they have distinct differences. One-class classification focuses on creating a model that learns to recognize instances from a single class, typically the "normal" or "benign" class and treats any deviation from this class as an anomaly. This approach is particularly useful when there is a lack of labeled data for the "anomalous" or "malicious" class, as it allows the model to learn the characteristics of the normal class without the need for labeled examples of anomalies.

Anomaly detection, on the other hand, is a broader concept that encompasses various techniques for identifying patterns in data that do not conform to expected behavior. While one-class classification is a specific method within anomaly detection, other anomaly detection techniques may not rely on the concept of a single class and can use different strategies to identify outliers or unusual patterns.

In this study, the need for one-class classification arises particularly in scenarios such as IoT malware detection, where access to labeled malicious samples are limited, and the diversity of malware types makes it challenging to cover all possible variations in a training dataset. By focusing on learning the characteristics of benign data, one-class classification provides a robust solution that can generalize well to detect not only known types of malwares but also unknown variants that may emerge in the future. This ability to adapt to new threats without requiring extensive labeled data makes one-class classification a valuable approach for maintaining the security of IoT systems in a constantly evolving threat landscape.

**4.4 Evaluation metrics**

In the context of IoT malware detection using one-class classification, the evaluation of the model's performance is crucial to ensure its effectiveness in identifying anomalies. Commonly used metrics for this purpose are precision, recall and the F1 scores, which provide insights into the model's accuracy and sensitivity, respectively.

Precision measures the proportion of correctly identified positive instances (true positives) out of all instances classified as positive (true positives and false positives). In the context of malware detection, a high precision indicates that the model is accurate in identifying malicious samples, with fewer benign samples being incorrectly labeled as malicious. While precision is important, it is not the sole focus in this scenario, as the cost of missing a true malware sample (false negative) can be significantly higher than the cost of a false alarm (false positive).

Recall, also known as sensitivity or true positive rate, measures the proportion of actual positive instances (true positives) that are correctly identified by the model out of all actual positive instances (true positives and false negatives). In the context of IoT malware detection, recall is of paramount importance. A high recall means that the model can detect a large proportion of the malware samples, minimizing the risk of undetected threats that could compromise the security of IoT systems. Given the potentially severe consequences of missed malware detections, prioritizing recall is a strategic decision aimed at ensuring the highest level of security.

In practice, there is often a trade-off between precision and recall, especially in the context of anomaly detection and one-class classification. To address this trade-off, the F1 score can be used as a harmonic means of precision and recall, providing a balanced measure of the model's performance. However, in scenarios where the primary goal is to capture as many anomalies or malware samples as possible, recall is given precedence. This means that while striving for a high recall, some decrease in precision may be acceptable to ensure that the model is highly sensitive to potential threats.

In anomaly detection problems, especially in scenarios where the class distribution is imbalanced (i.e., the number of normal instances outweighs the number of anomalies), the Area Under the Receiver Operating Characteristic (AUC-ROC) curve can be a better metric than precision, recall, or the F1 score. The ROC curve plots the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings, providing a comprehensive view of the model's performance across all thresholds. This is useful in anomaly detection, where the cost of false negatives can be high. The AUC-ROC is less sensitive to class imbalance than precision, recall, or F1 score, as it evaluates the model's ability to distinguish between classes rather than its ability to correctly label instances. Therefore, a high AUC-ROC score could indicate that the model is capable of detecting anomalies with a low rate of false positives, which is often the primary goal in anomaly detection tasks.

In addition to the above metrics, the Precision-Recall (PR) curve is another important tool for evaluating the performance of a malware detection model, especially in the context of imbalanced datasets. The PR curve plots the precision (y-axis) against the recall (x-axis) at various threshold settings, providing a detailed view of the trade-off between precision and recall for different threshold values. Which perfectly fits the goal of achieving higher recall in this study. Unlike the ROC curve, which plots the true positive rate against the false positive rate, the PR curve is more informative for imbalanced datasets where the number of negative instances (benign samples) significantly outweighs the number of positive instances (malware samples).

In scenarios where the actual number of negatives is large, a model might produce a large number of false positives while still maintaining a small false positive rate (FPR), leading to a misleadingly high Area Under the ROC Curve (AUC-ROC) score. This can create a false sense of security, as the model may not be as effective as the AUC-ROC score suggests. In contrast, the Precision-Recall curve is more sensitive to the number of false positives, as precision directly incorporates the number of false positives into its calculation. Therefore, a drop in precision due to an increase in false positives is immediately visible in the PR curve.

For this reason, in the context of IoT malware detection, where the cost of false negatives is high and the dataset is often imbalanced for anomaly detection problems, the PR curve can provide a more realistic assessment of the model's performance. A high area under the Precision-Recall curve (AUC-PR) indicates that the model could achieve high precision while maintaining high recall, which is crucial for effectively detecting malware without overwhelming the system with false alarms. And the precision-recall curve isn’t just providing a more realistic performance evaluation. The area under the PR curve (AUCPR) can also be a more informative summary statistic than the traditional AUC.

**5 Results and Comparison**

In this section, we primarily focus on comparing the performance of multi-class training, which utilizes both benign and malicious unlabeled samples, with that of one-class training, which exclusively employs benign samples in the training dataset. We evaluate two distinct algorithms, Isolation Forest and Deep Autoencoder, across both training approaches. While One-Class SVM is theoretically capable of achieving superior results in anomaly detection tasks, its practical application is limited by its inability to efficiently handle large-scale datasets. This limitation is particularly pronounced for non-linear SVMs, rendering them impractical for datasets of the size considered in this study.

**5.1 Train-test split**

For the benign data, we allocate 40% for training, 30% for testing, 20% for validation, and the remaining 10% for tuning. In the one-class training scenario, the entire 40% of benign data is used as the training set. However, for multi-class training, we modify the training set by replacing 20% of the benign samples with malicious samples to achieve an 80:20 ratio between benign and malicious data.

The test sets for both training approaches maintain an 80:20 ratio of benign to malicious samples. For each of the three major attack types—PartOfAHorizontalPortScan, DDoS, and Okiru—we select an equal number of samples corresponding to 20% of the benign test samples from each attack type, resulting in four test sets for each attack category. The rest attack types, there are total of 4981 samples combined after removing duplicates, these samples are combined into one separate test set called Attack, and a small benign sample is selected to also create an 80:20 ratio for this test set.

**5.2 Isolation Forest**

One class could achieve a very good performance for anomaly detection problems over isolation forest. However, the speed makes it intolerable when dealing with a large data set, especially for the non-linear kernels. After reducing the dimensions, the size of the data set is still too big even for linear SVM. On the other hand, Isolation Forest is an algorithm specifically designed for anomaly detection. It is based on the principle of isolating anomalies using decision trees, which are constructed by randomly selecting features and splitting values. This approach allows Isolation Forest to handle large datasets efficiently, as the computational complexity is linear with the number of samples and logarithmic with the number of trees in the forest. Additionally, Isolation Forest is less sensitive to the dimensionality of the data, as each tree considers only a subset of features at each split. This characteristic enables Isolation Forest to maintain fast performance and produce decent results even when the dimensionality of the data is high, making it a more suitable choice for anomaly detection in large and high-dimensional datasets. The following table shows the precision, recall, f1 scores and time related performance for all 3 test sets with 2 different training strategies.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Training method | Training time | Testing set | Test time | Precision | Recall | F1 |
| multi-class training | 8m51.5 | ddos | 7m35.5s | 0.46 | 0.73 | 0.57 |
| okiru | 6m54.4s | 0.54 | 1 | 0.7 |
| portscan | 6m35.6s | 0.47 | 0.76 | 0.58 |
| one-class training | 6m6.1s | ddos | 5m11.1s | 0.84 | 1 | 0.91 |
| okiru | 5m11.9s | 0.84 | 1 | 0.91 |
| portscan | 5m8.5s | 0.81 | 0.83 | 0.82 |

Table

In the above table, the results using one-class training are significantly better than using multi-class training which means the training set contains both benign and malicious samples. Using one-class training method, both the DDoS test set and Okiru test set achieve 100% recall with 84% precision, which means all the anomalies are captured, the PartOfHorizontalPortScan test set also achieves 95% recall, with 81% precision, compared to the results using multi-class training that only the Okiru test set reaches 100% recall but with only 54% precision. The DDoS and PartOfHorizontalPortScan test sets achieved 73% and 76% recall, with 46% and 47% precision. The training times and testing times are also slightly faster when using one-class training compared to using multi-class training. The results prove that using one-class training is not only feasible but also achieves much better results but also more robust to different types of malware attack.

Because the test sets are imbalanced, precision, recall scores may not accurately represent the performance of the models. Below images are the AUC-ROC and AUC-PR curves for all the test cases. It is clear to see that in both AUC plots, the area under curves for one-class training are bigger than the multi-class training, which means the overall performance is better. And the gaps between curves are also smaller when using one-class training, the curve for DDoS and Okiru test sets are even overlapped, which proves the robustness of the model.

A graph of a line graph

Description automatically generated with medium confidenceA graph of a line

Description automatically generated with medium confidence

Image (multi-class training)

A graph of a line graph

Description automatically generated with medium confidenceA diagram of a number of lines

Description automatically generated with medium confidence

Image (one-class training)

**5.3 Deep Learning Autoencoder**

In the previous tests for isolation forests, it has already been proven that one-class training with only benign samples is feasible and optimal. However, not all test sets have achieved 100% recall, and the precisions also have room to improve. Deep learning autoencoders can achieve better performance for anomaly detection compared to Isolation Forest, especially when dealing with large and complex datasets. Autoencoders can learn non-linear and high-dimensional representations makes them particularly effective for anomaly detection, as they can capture the underlying structure of normal data and identify deviations or anomalies more accurately. Furthermore, autoencoders can scale well with large datasets due to the parallel processing capabilities of modern deep learning frameworks and hardware accelerators like GPUs. In contrast, while Isolation Forest is efficient and effective for many anomaly detection tasks, it may not capture the complex dependencies and patterns in the data as well as autoencoders, particularly in high-dimensional spaces. Below table shows the performance metrics for 3 test sets using the same one-class training set.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Training time | Testing set | Test time | Percision | Recall | F1 |
| 5m46s \*  17 epochs | ddos0 | 8m58s | 0.9 | 1 | 0.95 |
| okiru0 | 9m15s | 0.9 | 1 | 0.95 |
| port0 | 9m13s | 0.9 | 1 | 0.95 |

Table

It is easy to see that using autoencoder, all 3 test sets have achieved the same results which is 100% recall and 90% precision. Not only did the performance improve compared to using the isolation forest model, but it has also achieved perfect robustness over 3 different types of attacks. Speed wise, the autoencoder was trained for 20 epochs but it stops converging at 17 epochs, and it takes 5m46s for each epoch. Considering the training is not designed to be performed on any IoT devices, and the improvement on performance, the longer training time is acceptable. The testing times are also longer than using isolation forest, for each sample it takes an average 41.13 microseconds compared to 23.11 microseconds using isolation forest model. The AUC plots below also indicates that the area are bigger than the isolation forest plots, and all 3 curves are perfectly aligned that indicate the robustness of the model.

A graph of a positive rate

Description automatically generated with medium confidenceA green line with black lines

Description automatically generated

Image (One-class autoencoder)

Because the autoencoder detects anomalies by using the reconstruction error, it could also be helpful to visualize the distribution of the errors. Below images display the distribution of reconstruction loss for all 3 test cases. The reconstruction loss for benign and malicious samples are well separated. In the Okiru and PartOfHorizontalPortScan test sets, there are overlapping situations for benign and malicious samples. And for all 3 test cases, there is a small part of benign falls between 0.125 and 0.150 on the x-axis, which will be misclassified as false positives.

A graph of a loss

Description automatically generated with medium confidence

A graph with numbers and a bar

Description automatically generated

A graph of a graph

Description automatically generated with medium confidence

Image (mse distribution)

**6 Conclusion and Future Work**

The findings from our research suggest a significant advancement in the domain of malware detection, favoring a one-class training approach over a multi-class training approach. Our expirements reveal that one-class training is not only feasible but also yields superior results compared to its multi-class counterpart. This success is primarily due to the one-class approach's inherent capacity to mitigate the training bias associated with specific types of malware data. Such biases often skew the learning process and compromise the model's generalizability. Additionally, one-class training offers a more economical and streamlined method for detecting malware by obviating the need for extensive malicious data collection and exhaustive labeling efforts. The robust performance of the one-class models, despite the reduced requirement for labeled data, marks a promising direction for efficient and cost-effective malware detection strategies.

Future research endeavors will concentrate on refining the data foundation, feature representation, and model architecture to bolster the effectiveness of malware detection systems further. The next phase will seek to incorporate data that mirrors the unpredictability and complexity of real-world scenarios more closely, thus addressing the limitations of data collected from controlled environments. Emphasis will also be placed on enhancing feature extraction techniques, particularly using n-grams within connection history analysis. This shift aims to capture a broader spectrum of behavioral patterns, though it may result in an increased number of features. Therefore, feature selection and reduction techniques will be essential to manage the expanded feature set without compromising the model's interpretability or performance.

In parallel, efforts will be made to explore the potential of more intricate autoencoder designs. The relatively basic autoencoder employed in this study offers a foundational starting point, suggesting a vast unexplored potential for complex architectures. Innovations in autoencoder structures, potentially integrating advanced neural network techniques, hold the promise of significantly improving the model's capability to detect subtle and sophisticated patterns indicative of malware presence.

**7 References**

[1] Pajouh, H. H., Dehghantanha, A., Khayami, R., & Choo, K.-K. R. (2018). Intelligent OS X malware threat detection with code inspection. Journal of Computer Virology and Hacking Techniques, 14(3), 213-223.

[2] Sudheera, Kalupahana Liyanage Kushan, et al. "ADEPT: Detection and Identification of Correlated Attack Stages in IoT Networks." IEEE Internet of Things Journal 8.8 (2021): 6591-6607

[3] Sahu, Amiya Kumar, et al. "Internet of Things attack detection using hybrid Deep Learning Model." Computer Communications (2021).

[4] Fang, Z., Wang, J., Li, B., Wu, S., Zhou, Y., & Huang, H. (2019). Evading Anti-Malware Engines with Deep Reinforcement Learning. IEEE Access, 7, 48867-48879.

[5] Jahromi, A. N., Hashemi, S., Dehghantanha, A., Parizi, R., & Choo, K. (2020). An Enhanced Stacked LSTM Method With No Random Initialization for Malware Threat Hunting in Safety and Time-Critical Systems. IEEE Transactions on Emerging Topics in Computational Intelligence, 4(5), 630-640.

[6] Pei, X.-j., Deng, X., Tian, S., Zhang, L., & Xue, K. (2023). A Knowledge Transfer-based Semi-Supervised Federated Learning for IoT Malware Detection. IEEE Transactions on Dependable and Secure Computing, 20, 2127-2143.

[7] Tajoddin, A., Abadi, M. RAMD: registry-based anomaly malware detection using one-class ensemble classifiers. Appl Intell 49, 2641–2658 (2019).

[8] Al-Qudah, M., Ashi, Z., Alnabhan, M. M., & Abu Al-haija, Q. (2023). Effective One-Class Classifier Model for Memory Dump Malware Detection. J. Sens. Actuator Networks.

[9] Sebastian Garcia, Agustin Parmisano, & Maria Jose Erquiaga. (2020). IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]. Zenodo.

[10] Shao, Enchun (2019). Encoding IP Address as a Feature for Network Intrusion Detection. Purdue University Graduate School. Thesis.

[11] Ye, Y., Chen, L., Hou, S., Hardy, W., & Li, X. (2018). DeepAM: A heterogeneous deep learning framework for intelligent malware detection. Knowledge and Information Systems, 54, 265-285.

[12] Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., & Chen, J. (2018). Detection of Malicious Code Variants Based on Deep Learning. IEEE Transactions on Industrial Informatics, 14, 3187-3196.

[13] Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., & Venkatraman, S. (2019). Robust Intelligent Malware Detection Using Deep Learning. IEEE Access, 7, 46717-46738.

[14] Pu, G., Wang, L., Shen, J., & Dong, F. (2021). A hybrid unsupervised clustering-based anomaly detection method. Tsinghua Science and Technology.

[15] Zhang, L., Yin, J., Ning, J., Wang, Y., Adebisi, B., & Yang, J. (2022). A Novel Unsupervised Malware Detection Method based on Adversarial Auto-encoder and Deep Clustering. 2022 9th International Conference on Dependable Systems and Their Applications (DSA), 224-229.