INL/RPT-24-04922 Rev:000

Summer Internship Report

Advanced Data Science Model for Detecting Intelligent Malware

July 2024

Allan Aanonsen
Intern, Irradiation Experiments and Thermal Hydraulics Analysis,
Idaho National Laboratory

Palash Kumar Bhowmik
Mentor, Thermal Fluid Systems Methods and Analysis
Idaho National Laboratory

Guenevere Chen
Advisor, Electrical and Computer Engineering,
University of Texas at San Antonio

Piyush Sabharwall
Manager, Thermal Fluid Systems Methods and Analysis
Idaho National Laboratory

Idaho National Laboratory

INL is a U.S. Department of Energy National Laboratory
operated by Battelle Energy Alliance, LLC

INL/RPT-24-04922 Rev:000

Summer Internship Report

Advanced Data Science Model for Detecting Intelligent Malware

Allan Aanonsen
Thermal Fluid Systems Methods and Analysis,
Idaho National Laboratory

Palash Kumar Bhowmik
Mentor, Thermal Fluid Systems Methods and Analysis,
Idaho National Laboratory

Guenevere Chen
Advisor, Electrical and Computer Engineering,
University of Texas at San Antonio

Piyush Sabharwall
Manager, Thermal Fluid Systems Methods and Analysis,
Idaho National Laboratory

July 2024

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

ABSTRACT

This study focused on developing a robust artificial intelligence (AI) model capable of detecting and characterizing advanced malware in Internet of Things (IoT) devices using network data. By analyzing network traffic with various machine learning (ML) models, our AI model can identify and characterize malicious activities to significantly improve malware detection accuracy and reliability as compared to traditional methods. The developed AI/ML model was trained using network data from IoT devices, leveraging classifiers such as Random Forest, Gradient Boosting, AdaBoost, and others to optimize detection performance. This project demonstrates a scalable framework for real-time malware detection and characterization in IoT networks, capable of identifying infected devices and facilitating the necessary steps to remove or isolate them, thereby preventing further infections. Although digital twin (DT) integration is not yet implemented in the current model, it represents a promising future enhancement. By creating a virtual replica of physical IoT devices, DT technology would allow for real-time monitoring and analysis without directly accessing operational technology, thus reducing the risk of compromising or reducing the performance of actual devices. This integration would further enhance the security of IoT ecosystems, combining AI technology to better flag and detect indications of malware-infected devices within a nuclear system environment.

Page intentionally left blank

ACKNOWLEDGEMENTS

The author would like to thank the Thermal Fluid Systems Methods and Analysis department in the Reactor System Design and Analysis Division at Idaho National Laboratory (INL) for their encouragement and support. The author would also like to express his gratitude to the Department of Electrical and Computer Engineering at University of Texas at San Antonio (UTSA) for providing their support during this INL internship. Appreciation also goes out to P. K. Bhowmik and P. Sabharwall at INL and G. Chen at UTSA for their mentoring, advising, and guidance.

Page intentionally left blank

CONTENTS

FIGURES

TABLES

ACRONYMS

AI	artificial intelligence
C&C	command and control
CNN	convolution neural network
DBF	Deep Blockchain Framework
DDoS	distributed denial of service
DL	deep learning
DNN	deep neural network
DT	digital twin
FN	False Negatives
IAM	Identify and Access Management
IoT	Internet of Things
IP	Internet protocol
LSTM	long short-term memory
ML	machine learning
OT	operational technology
PCA	Principal Component Analysis
RNN	recurrent neural network
SCADA	supervisory control and data acquisition
TP	True Positives

Page intentionally left blank

Advanced Data Science Model for Detecting Intelligent Malware

1. INTRODUCTION

1.1 Background and Motivation

The rapid growth of Internet of Things (IoT) devices connected to the Internet has introduced new vulnerabilities, enabling sophisticated malware attacks that threaten the security and reliability of IoT ecosystems. Traditional methods of malware detection often struggle to keep up with the evolving nature of these threats. However, artificial intelligence (AI) models can significantly improve malware detection, accuracy, and reliability. By leveraging AI technology, we can combine the computational power of machine-learning (ML) algorithms with human expertise to better flag and detect indications of possible cyber-attacks. In this study, the AI model analyzes data collected from network traffic of infected IoT devices to identify potential malware threats. This approach not only enhances the reliability of malware detection but also helps maintain the performance and safety of the IoT ecosystem by providing timely and accurate threat identification.

1.2 Objectives of the Study

The primary objective of this study was to develop a robust AI model capable of detecting and characterizing malware on IoT devices through network traffic analysis. Given the unique vulnerabilities and security challenges presented in IoT devices, the study aimed to achieve the following objectives:

1. **Enhance Malware Detection Accuracy:**
   - Develop an AI-based solution that significantly improves the accuracy of malware detection in IoT devices as compared to traditional methods.
   - Utilize advanced ML techniques to analyze network traffic and accurately identify malicious activities.
2. **Evaluation and optimization of ML Models:**
   - Explore and evaluate the performance of various ML classifiers, including Random Forest, Gradient Boosting, Ada Boost, Logistic Regression, Decision Tree, Light GBM, and Extra Trees.
3. **Save and Deploy Model Components Securely:**
   - Save trained model components, including scaler parameters and model weights as .npy files to facilitate secure and easy deployment.
   - Ensure reliability and trustworthiness by building a Docker container for the AI model.
4. **Lay the foundation for Future Integration with DT Technology:**
   - Design the methodology to be extensible, allowing for future integration with digital twin (DT) technologies.
   - Highlight the potential of DTs to create virtual replicas of IoT devices for secure and efficient monitoring without direct access to operational technology (OT).

1.3 Literature Review and Scope of the Study

This study focuses on developing an algorithm that leverages the pre-analyzed and labeled IoT-23 dataset to detect and characterize malware on IoT devices. Modern malware has become increasingly sophisticated and adept at evading detection. However, regardless of its concealment strategies, malware must execute its functions, which inevitably generate detectable patterns in network activity. By using the labeled data from the IoT-23 dataset, we can identify these patterns more accurately.

A summary of the selected available studies with the detection types, method/tools used, and performance matrix are listed in Table 1.

Table 1. Deep Learning for Cyber-Threat Detection in IoT Networks.

Author	Detection Types	Method/Tools Used	Performance Matrix
Fatani et al., 2021 [1]		DL/TSODE	Multi-classification: BoT-IoT = 99.04%, KDDCup-99 =92.06%, NSL-KDD = 75.75%, and CICIDS-2017 = 99.93%.
Nie et al., 2020 [2]		DL	Acc = 97.60%.
Ullah and Mahmoud, 2021 [3]		DL/CNN1D, CNN2D, and CNN3D	Multiclass classification: CNN1D= 99.97%, CNN2D= 99.95%, and CNN3D =99.94%.
Liang et al., 2021 [4]		DL/OICS-VFSL	NSL-KDD = 91%.
Muthanna et al., 2022 [5]		DL/Cu-LSTM-GRU	Cu-LSTM-GRU = 99.23%.
Zeeshan et al., 2021 [6]		DL/PB-DID	Acc = 96.3%.
Qiu et al., 2021 [7]		DL/ID-Based NIDS	Acc = 94.31%.
Ibitoye et al., 2019 [8]		DL/FNN IDS and SNN IDS	FNN ≈ 51 % and SNN ≈ 51%.
Mehedi et al., 2022 [9]		DL/P-ResNet	Acc = 87%.
Zhou et al., 2021 [10]		DL/HAA	Precision ≈ 30%.
Wahab et al., 2022 [11]		DL/Principal Component Analysis (PCA)	Acc ≈ 99.23%.
Alkadi et al., 2020 [12]	RNN & LSTM	Hybrid DL/Deep Blockchain Framework (DBF)	UNSW-NB15 = 97.26 % and BoT-IoT = 96.71.
Taher et al., 2022 [13]	RNN & LSTM	Hybrid DL/TSA-LSTMRNN	Acc = 92.67%.
Abdel-Basset et al., 2021 [14]	RNN & LSTM	Hybrid DL/SS-Deep-ID	Binary classification: CIC-IDS2017 = 99.6% and CIC-IDS2018 = 99.33%.
Alani et al., 2022 [15]	RNN/LSTM & CNN	Hybrid DL/Two-layer IoT IDS	Acc = 99.15%.

Author	Detection Types	Method/Tools Used	Performance Matrix
Thamilarasu et al., 2019 [16]	DBN & DNN	Hybrid DL/IIDS	Precision rate = 95%.
Zhang et al., 2019 [17]	DBN & DNN	Hybrid DL/GA-DBN	Acc ≈ 98.82%.
De Elias et al., 2022 [18]	CNN & LSTM	Hybrid DL/Improved hybrid CNN-LSTM	Binary classification = 97.85% Multiclass classification = 97.14%.
Kacha et al., 2022 [19]	CNN & LSTM	Hybrid DL/Hybrid CNN-LSTM	Binary classification = 100% Multiclass classification = 98.69%.
Jahromi et al., 2023 [20] Ahmad et al., 2022 [21]	DNN & DT	Hybrid DL/Cyber-Threat Hunting	Secure Water Treatment (SWaT): Centralized = 95.15% and Federated = 95.12%. Gas pipeline dataset: Centralized = 99.64% and Federated =99.66%
Al-Hamadi et al., 2020 [22]		ML/ADIoTS	N/A.
Eskander et al., 2020 [23]		ML/Passban	iForest =96.6% and LOF =88.25%.
Abu Al-Haija et al., 2020 [24]		DL/IoT-IDCS-CNN	Acc = 98.2–99.3% (98.75% average).
Gad et al., 2021 [25]		ML/VANETs	Binary classification = 98.2% Multi-classification = 97.9%
Shafiq et al., 2020 [26]		ML/CorrAUC	Acc = 99.99%.
Gao et al., 2022 [27]		ML/BSBC-RF	Acc = 99.96%.
Nie et al., 2021 [28]		ML/DDPG	
Tina Rezaei et al., 2021 [29]	DNN	Neural Network embedding/K-means clustering	BatchSize =30, Accuracy =97.75%, Precision=98.44%, Recall=97.04%, and Fmeasure=97.73%.

Note: Here, CNN for convolution neural network, DL for deep learning, DNN for deep neural network, LSTM for long short-term memory, ML for machine learning, RNN for recurrent neural network.

The scope of this study includes:

- **Dataset Utilization:** Utilizing the IoT-23 dataset, which contains pre-labeled network traffic data, to train and test various ML classifiers. This dataset provides a comprehensive foundation for detecting malicious behaviors.
- **Algorithm Development:** Developing an algorithm that applies advanced ML techniques to scan the IoT-23 dataset. The algorithm trains on different classifiers, such as Random Forest, Gradient Boosting, AdaBoost, Logistic Regression, Decision Tree, Light GBM, and Extra Trees, to identify which classifier performs the best. The algorithm must be reliable, explainable, and trustworthy.
- **Model Optimization:** Performing hyperparameter tuning using GridSearchCV to optimize each classifier for the best performance.
- **Network Feature Importance:** Identifying the importance of different network features during the training of each classifier. This involves determining which features are most significant for each classifier in accurately detecting malware.
- **Model Saving Security:** Saving the model weight in a secure way to avoid model serialization attacks.

By focusing on these areas, the study aims to develop robust techniques for detecting and characterizing malware, thereby enhancing the security and reliability of IoT ecosystems.

1.4 Methodology

The methodology for this project involves several key steps in developing a model for malware detection in IoT devices. The steps are outlined as follows:

1. **Data Collection:**
    - **Dataset Selection:** The IoT-23 dataset, which consists of network traffic captures from various IoT devices, was selected for this study. This dataset includes pre-labeled data representing both malicious and benign activities, providing a rich source of information for training and testing ML models.
    - **Data Loading:** All relevant data files from the IoT-23 dataset were loaded into a unified framework for further processing.
2. **Data Preprocessing:**
    - **Data Cleaning:** The raw data was cleaned to handle missing values, remove duplicates, and address any inconsistencies. Invalid values were replaced with 0, and missing values were filled in with -1 to ensure data integrity.
    - **Data Integration:** Multiple conn.log files from different scenarios were combined into a single comprehensive dataset. This integration included standardizing column names and ensuring consistent labeling across the dataset.
    - **Label Standardization:** Different representations of the same labels were unified to maintain consistency throughout the dataset. For instance, variations of benign and malicious labels were standardized.
    - **Feature Encoding:** Categorical features such as proto and conn_state were converted into numerical values using one-hot encoding. This step was performed using pd.get_dummies() to create binary columns for each category, facilitating their use in the ML models.

3. **Feature Scaling:**
    - **Standardization:** Standardizing numerical features is a common preprocessing step in ML, which ensures that each feature contributes equally to the model's performance, preventing features with larger magnitudes from dominating the learning process. This standardization can lead to better model convergence and performance.
4. **Model Training and Evaluation:**
    - **Classifier Training:** Various ML classifiers, including Random Forest, Gradient Boosting, AdaBoost, Logistic Regression, Decision Tree, Light GBM, and Extra Trees, were trained on the dataset.
    - **Hyperparameter Tuning:** Hyperparameter tuning was performed using GridSearchCV to optimize each classifier for the best performance.
    - **Model Evaluation:** The performance of each classifier was evaluated using metrics such as accuracy, precision, recall, and F1-score. Confusion matrices were generated to understand the classification results in detail.
5. **Network Feature Importance:**
    - **Feature Analysis:** During the training process, the importance of different network features was identified for each classifier. This analysis highlighted which features were most significant in detecting malware, such as protocol types, data volumes, and connection states.
    - **Visualization:** Feature importance was visualized using bar plots to provide insights into which features were most influential in the classification process.
6. **Model Deployment:**
    - **Docker Container:** A Docker container was created to encapsulate the entire environment, including the AI model, dependencies, and necessary runtime configurations. This containerization approach ensures consistent performance across different deployment platforms and enhances the reliability and security of the solution.
7. **Model Saving and security measures:**
    - **Model Saving:** Trained model components, including scaler parameters and model weights, were saved as .npy files. This approach ensures secure and easy deployment of the models.
    - **Security Measures:** Additional security measures were investigated to protect the models from potential threats, such as model serialization attacks. Methods such as authenticated access, encryption, and checksum verification are to be added in the future once a more reliable and more accurate model has been developed.
8. **Future Integration and Application:**
    - **Digital Twin Technology:** The methodology was designed to be extensible, allowing for future integration with DT technologies. This would enable the creation of virtual replicas of IoT devices for secure and efficient monitoring without direct access to OT.
    - **Real-World Application:** While not implemented in the current study, future work will focus on applying these detection methods to real-world IoT networks to validate their effectiveness and reliability in identifying infected devices.

By following these steps, the study aims to develop a comprehensive and effective AI-based solution for malware detection in IoT devices, contributing to enhanced security and reliability in IoT ecosystems. The model will be explainable, trustworthy, and reliable.

1.5 Approach

Figure 1 shows the approach methodology taken in this study.

Figure 1. Approach methodology.

2. TARGETED MALWARE

2.1 Botnet Overview

Figure 2 shows a graphic of different types of malware. One serious type of malware is a botnet. The term "botnet" is a combination of the terms "bot" and "net." The bot part refers to the ability of the malware to automate things and tasks like a robot would. The second part refers to a network of compromised devices. A botnet is a form of malware that attacks computers on the Internet and controls them with command and control (C&C) servers to perform a wide variety of automated tasks, which include things like sending spam emails and performing distributed denial of service (DDoS) attacks. One of the most notable botnets in previous years was the Mirai botnet, which hit millions of online devices—especially IoT devices—by scanning and identifying vulnerable machines and taking advantage of those left with default login credentials [30]. Here are some of the defining characteristics of botnets:

- **Tasks Performed by Botnets:**
  1. Advertising fraud and sending spam emails.
  2. Cryptocurrency mining.
  3. Stealing personal data and sensitive information.
  4. Performing brute force attacks.
- **Phases of Botnets:**
  1. **Infections:** In this phase, the attackers infect the targeted machines by sending the malware.
  2. **Connection:** In this phase, the botnet initiates an internet connection.
  3. **Control:** In this phase, the attack occurs, for example, sending spam emails.
  4. **Multiplication:** In this phase, the botnet will try to compromise more machines to join them in the network and become what we call zombies.

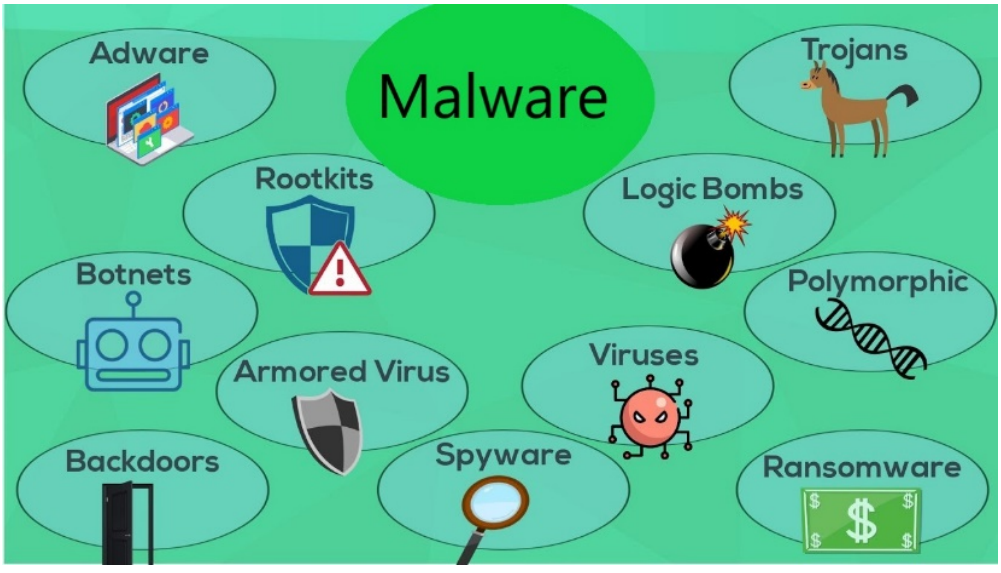

Figure 2. Types of malware.

2.2 Horizontal Scans

Horizontal scans are a type of network reconnaissance attack where an attacker systematically scans multiple Internet protocol (IP) addresses for open ports on the same port number. This technique is used to gather information about potential vulnerabilities in a network, which can then be exploited in subsequent attacks. Horizontal scanning involves sending packets to various IP addresses within a network to check for open ports. Unlike vertical scanning, which targets multiple ports on a single IP address, horizontal scanning focuses on one port across many IP addresses. This method helps attackers identify common services or vulnerabilities running on specific ports. Attackers often use horizontal scans to find IoT devices with open ports that still use default credentials. Once identified, these devices can be compromised and added to a botnet, as described below:

- **Tasks Performed by Horizontal Scans:**
  1. Identify vulnerable services.
  2. Map the network.
  3. Exploit known vulnerabilities.
  4. Gather intelligence for targeted attacks.
- **Phases of Horizontal Scans:**
  1. **Reconnaissance:**
     - In this phase, attackers gather preliminary information about the target network, such as IP address ranges and active hosts.
  2. **Scanning:**
     - Attackers send packets to a specific port number across multiple IP addresses. They monitor the responses to determine which IP addresses have the port open and possibly what service is running on it.
  3. **Analysis:**
     - The data collected from the scanning phase is analyzed to identify potential vulnerabilities. Attackers look for patterns, such as the same service running on many hosts, which could indicate a common vulnerability.
  4. **Exploitation:**
     - Using the information from the analysis phase, attackers can exploit identified vulnerabilities to gain unauthorized access, execute code, or disrupt services.

3. DATA COLLECTION

The IoT-23 dataset was used for this analysis, which consisted of the network traffic captures from IoT devices. The dataset includes 20 malware captures executed on IoT devices that have been collected and labeled. Each scenario was provided in the form of a conn.log file that was generated by the Zeek network analyzer. With this dataset, we aim to enhance the accuracy and reliability of malware detection on IoT devices with access to network resources.

To prepare the data for analysis, the following steps were performed [31]:

- The 20 malicious conn.log-labeled captures corresponding to the various scenarios of the IoT-23 dataset were loaded. These files contain the network traffic data that were labeled for malicious and benign activities.
- The column names were standardized and the data from all the loaded files were combined into a single data frame. Unnecessary columns such as ts, uid, id.orig_h, id.resp_h, id.resp_p, service, local_orig, local_resp, and history were dropped.
- The labels were standardized to ensure consistency across the dataset since different representations of the same label were unified.
- Any invalid values were replaced with 0 and missing values were filled in with -1 to ensure data consistency.
- Categorical features were converted into numerical values using one-hot encoding.
- Once the data was preprocessed, it then was saved into a single .csv file for further analysis and model training.

Figure 3 shows the distribution of the anomalies that were discovered.

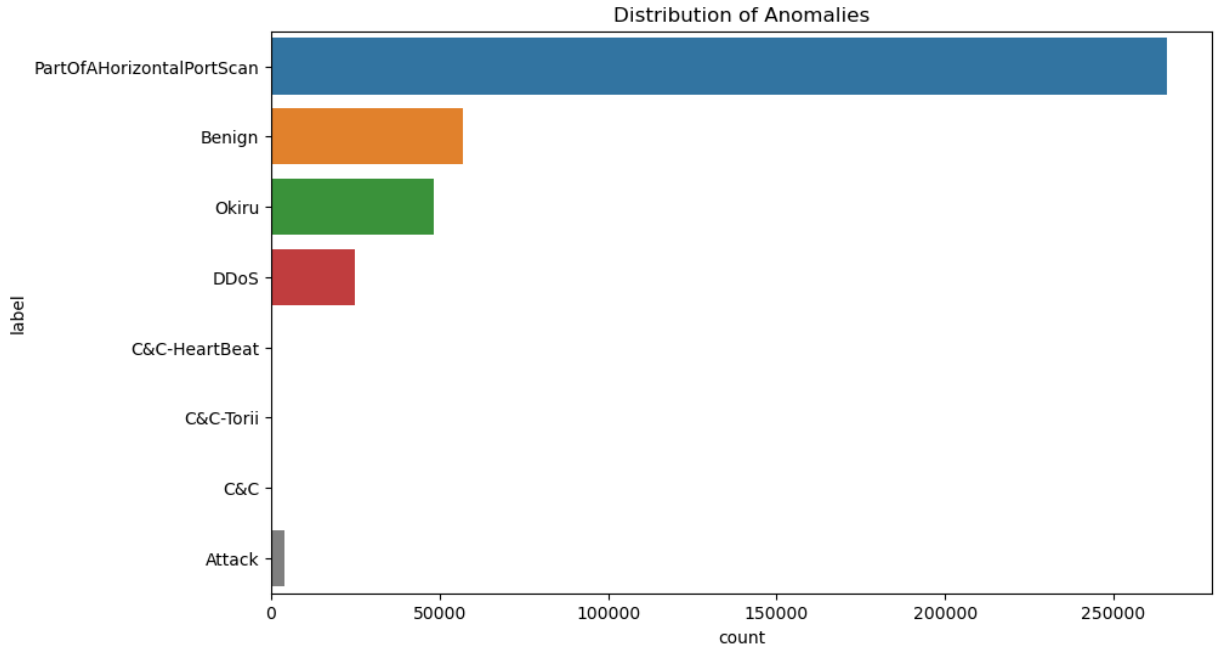

Figure 3. Distribution of anomalies.

3.1 Label Meaning Used for Classification

This section gives a detailed explanation of the malware dataset labels used in this analysis with descriptions from the IoT-23 dataset page and how they were labeled. Understanding these labels will help interpret the performance metrics of each classifier in this report [32]:

- **Attack:** Some type of attack occurred from an infected device to another host. This is labeled as an attack to any flow that, by analyzing its payload and behavior, tries to take advantage of a vulnerable service (a brute force to some telnet login, a command injection in the header of a GET request, etc.).
- **Benign:** No suspicious or malicious activities were found in the connections.
- **C&C:** The infected device was connected to a C&C server. This activity was detected in the analysis of the network malware capture due to periodic connections to the suspicious server where the infected device appears to be downloading some binaries or decoded orders are sent and received.
- **DDoS:** A DDoS attack is being executed by the infected device. These traffic flows are detected as part of a DDoS attack due to the large number of flows being directed to the same IP address.
- **HeartBeat:** Packets sent on this connection are used to keep track of the infected host by the C&C server. HeartBeat was detected by filtering connections with response bytes lower than 1B and with periodic similar connections. Normally, this is combined with some known suspicious destination port or destination IP known to be a C&C server.
- **Mirai:** The connections seem to have the same characteristics as a Mirai botnet. This label is added when the flows have a similar pattern as the most commonly known Mirai attacks.
- **Okiru:** The connections have the characteristics of an Okiru botnet. This label is added when the flows have a similar pattern to Okiru attacks, which are less common than Mirai attacks, but do occur.
- **PartOfAHorizontalPortScan:** The connections are used to initiate a horizontal port scan to gather information to perform further attacks. These labels were used when a similar number of transmitted bytes and multiple different destination IPs followed the same pattern from the same port.
- **Torii:** These connections have the same characteristics as a Torii botnet. This label is added when the flows have a similar pattern to Torii attacks, which are less common than Mirai attacks, but do occur.

3.2 Data Splitting

A random splitting strategy then was employed to ensure the dataset was divided into training and testing sets in an unbiased manner. This approach helps in evaluating the model's performance on unseen data and ensures the model generalizes well to new, unseen instances. The data was split as follows:

- **Training Set:** 80% of the data was allocated for ML model training. This set was used to learn the patterns and relationships within the data.
- **Testing Set:** 20% of the data was reserved for testing the models. This set was used to evaluate the performance and accuracy of the models on the unseen data.

The details of the split are as follows:

- **Training Features Shape:** The training set consisted of 320,000 instances and 24 features. This large number of instances ensures the model has enough data to learn from, which is crucial for capturing underlying patterns and improving model accuracy.
- **Testing Features Shape:** The testing set consists of 80,000 instances and 24 features. This substantial testing set provides a robust evaluation of the model's performance, helping to identify any issues with overfitting or underfitting.

4. RESULTS AND DISCUSSION

4.1 Model Accuracy

The evaluation results of various classifiers highlight their effectiveness in terms of accuracy for malware detection and classification. Each classifier achieved an accuracy rate of approximately 80%. These results demonstrate that all models performed fairly well in terms of accuracy, making them a strong candidate for robust malware detection and classification. Figure 4 shows the classifier accuracy breakdown, while Table 2 shows a summary of the model accuracy.

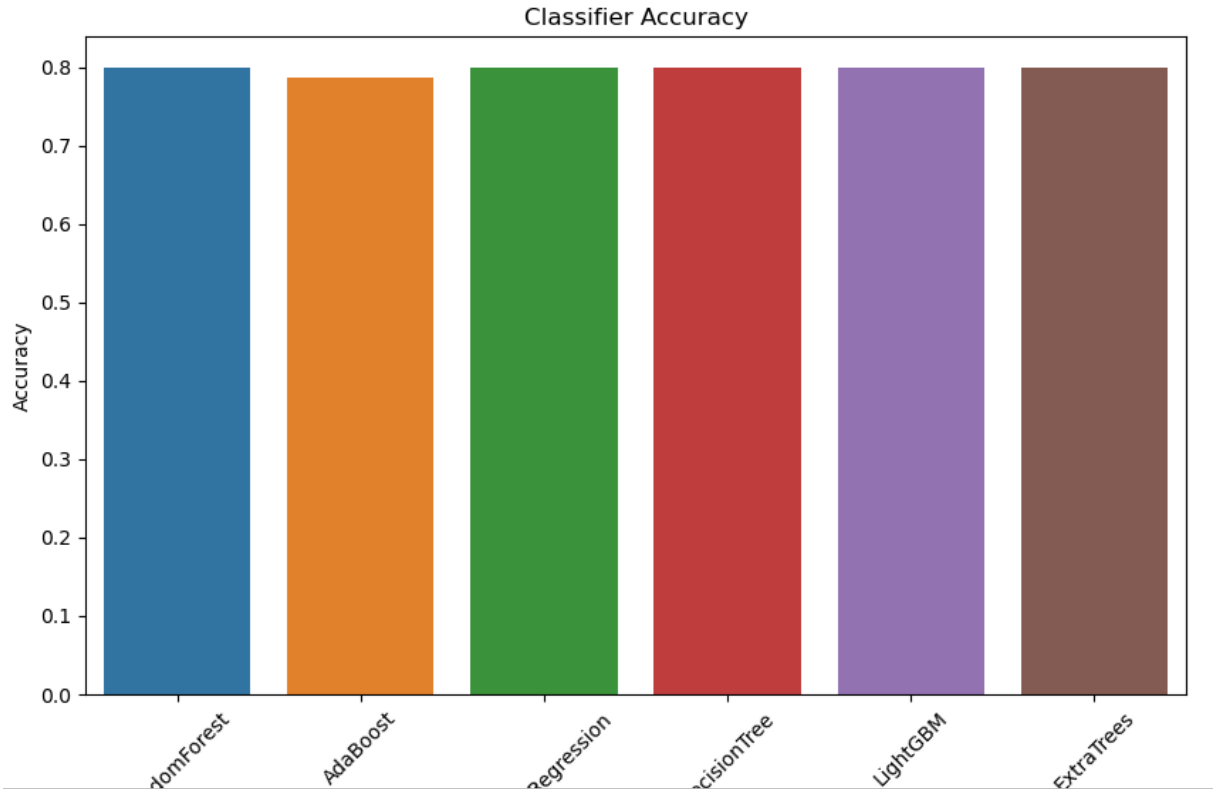

Figure 4. Classifier accuracy.

Table 2. Model Accuracy Summary.

Model	Accuracy	Time Cost
Random Forest	0.799750	115.30 seconds
Gradient Boosting	0.794863	1270.37 seconds
Ada Boost	0.787200	354.44 seconds
Logistic Regression	0.799562	266.65 seconds
Decision Tree	0.799763	8.90 seconds
Light GBM	0.799813	168.34 seconds
Extra Trees	0.799900	69.32 seconds

4.2 Best Parameter Found for Each Classifier using GridSearchCV

GridSearchCV was employed to find the best parameters for the optimization of each classifier. This technique systematically works through multiple combinations of parameter values, cross-validating as it goes to determine which combination provides the best performance, as shown in Table 3. The best parameters for each classifier, as well as a definition of what they look for, are provided below:

- **Random Forest Parameters:**
  - n_estimators: The number of trees in the forest.
  - max_depth: The maximum depth of the tree. If none, then the nodes are expanded until all leaves are pure or until all leaves contain less than the min_samples_split samples.
- **Gradient Boosting Parameters:**
  - learning_rate: Learning rate shrinks the contribution of each tree by learning_rate. There is a trade-off between learning_rate and n_estimators. Values must be in the range of [0.0, inf].
  - n_estimators: The number of boosting stages to perform. Gradient boosting is fairly robust to overfitting, so a large number usually results in better performance. Values must be in the range of [1, inf].
- **Ada Boost Parameters:**
  - learning_rate: Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the learning_rate and the n_estimators parameters. Values must be in the range of [0.0, inf].
  - n_estimators: The maximum number of estimators at which boosting is terminated. In the case of a perfect fit, the learning procedure is stopped early. Values must be in the range of [1, inf].
- **Logistic Regression Parameters:**
  - C: Inverse of regularization strength; must be a positive float. As with support vector machines, smaller values specify stronger regularization.
- **Decision Tree Parameters:**
  - max_depth: The maximum depth of the tree. If none, then the nodes are expanded until all leaves are pure or until all leaves contain less than the min_samples_split samples.
- **Light GBM Parameters:**
  - learning_rate: Controls the contribution of each tree to the overall model, which is also known as the shrinkage parameter.
  - n_estimator: Specifies the number of boosting rounds or trees to be added to the model.
- **Extra Trees Parameters:**
  - max_depth: The maximum depth of the tree. If none, then the nodes are expanded until all leaves are pure or until all leaves contain less than the min_samples_split samples.
  - n_estimators: The number of trees in the forest.

Table 3. GridSearchCV parameters.

Model	Best Parameters	Total Number of Fits
Random Forest	n_estimators: 50 max_depth: 10	24
Gradient Boosting	learning_rate: 0.1 n_estimators: 100	24
Ada Boost	learning_rate: 0.1 n_estimators: 50	24
Logistic Regression	C: 1	16
Decision Tree	max_depth: 10	12
Light GBM	learning_rate: 0.01 n_estimators: 100	24
Extra Trees	max_depth: 20 n_estimators: 100	24

These parameters were determined to provide the optimal balance between bias and variance, thereby improving the performance and robustness of each classifier. By fine-tuning these parameters, the models are not only ensured to be accurate but also capable of generalizing well to new, unseen data, which is crucial for effective malware detection and classification in DT technology. The total number of fits for each classifier reflects the number of parameter combinations evaluated during the GridSearchCV process, ensuring thorough exploration of the parameter space.

4.3 Classification Report for Each Classifier

This section provides a detailed explanation of the labels used in this malware dataset, their meanings, and their importance:

- **Precision** measures the accuracy of the positive predictions. It is the ratio of correctly predicted positive observations to the total predicted positive:

$$\text{Precision} =$$

- **Recall**, also known as Sensitivity or True Positive Rate, measures the ability of the classifier to find all positive samples. It is the ratio of correctly predicted positive observations to all observations in the actual class:

$$\text{Recall} =$$

- **F1-score** is the harmonic mean of precision and recall. It balances the two metrics and is useful when it becomes necessary to take both false positives and false negatives into account. It provides a single metric to capture both precision and recall:

$$\text{F1-score} =$$

- **Support** is the number of actual instances of that class in the dataset. This helps in understanding the distribution of the dataset, which is crucial for interpreting performance metrics. For instance, a high precision or recall for a class with very few instances, such as C&C with a value of 3, might not be as meaningful as for a class with many instances such as PartOfAHorizontalPortScan, which has a value of 53,191.

Table 4 through Table 10 provide the classification reports for the parameters used in this study.

Table 4. Random forest classification report.

Categories	Precision	Recall	F1-score	Support

Attack	1.00	0.99	1.00	775
Benign	0.99	0.88	0.93	11534
C&C	0.00	0.00	0.00	3
C&C-Heartbeat	0.50	1.00	0.67	2
C&C-Torii	1.00	0.25	0.40	8
DDoS	0.00	0.00	0.00	4946
Okiru	0.50	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

Table 5. Gradient boosting classification report.

Categories	Precision	Recall	F1-score	Support
Attack	0.79	0.70	0.74	775
Benign	0.96	0.88	0.92	11534
C&C	0.00	0.00	0.00	3
C&C-Heartbeat	0.00	0.00	0.00	2
C&C-Torii	0.00	0.00	0.00	8
DDoS	0.50	0.00	0.00	4946
Okiru	0.50	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

Table 6. Ada boost classification report.

Categories	Precision	Recall	F1-score	Support
Attack	0.00	0.00	0.00	775
Benign	0.93	0.85	0.89	11534
C&C	0.00	0.00	0.00	3
C&C-Heartbeat	0.00	0.00	0.00	2
C&C-Torii	0.00	0.00	0.00	8
DDoS	0.00	0.00	0.00	4946
Okiru	0.00	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

Table 7. Logistic regression classification report.

Categories	Precision	Recall	F1-score	Support
Attack	1.00	0.99	0.99	775
Benign	0.99	0.88	0.93	11534
C&C	0.00	0.00	0.00	3
C&C-Heartbeat	0.00	0.00	0.00	2
C&C-Torii	1.00	0.12	0.22	8
DDoS	0.00	0.00	0.00	4946
Okiru	0.00	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

Table 8. Decision tree classification report.

Categories	Precision	Recall	F1-score	Support
Attack	1.00	0.99	0.99	775
Benign	0.99	0.88	0.93	11534
C&C	0.00	0.00	0.00	3
C&C-Heartbeat	1.00	1.00	1.00	2
C&C-Torii	1.00	0.25	0.40	8
DDoS	0.00	0.00	0.00	4946
Okiru	0.50	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

Table 9. Light GBM classification report.

Categories	Precision	Recall	F1-score	Support
Attack	1.00	0.99	1.00	775
Benign	0.99	0.88	0.93	11534
C&C	1.00	0.33	0.50	3
C&C-Heartbeat	0.33	1.00	0.50	2
C&C-Torii	0.50	0.25	0.33	8
DDoS	0.00	0.00	0.00	4946
Okiru	0.50	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

Table 10. Extra trees classification report.

Categories	Precision	Recall	F1-score	Support
Attack	1.00	1.00	1.00	775
Benign	0.99	0.88	0.93	11534
C&C	1.00	0.33	0.50	3
C&C-Heartbeat	0.50	1.00	0.67	2
C&C-Torii	1.00	0.25	0.40	8
DDoS	0.00	0.00	0.00	4946
Okiru	0.50	0.00	0.00	9541
PartOfAHorizontalPortScan	0.77	1.00	0.87	53191

4.4 Classification Report Evaluations

The evaluation of various classifiers on the given dataset yielded insightful results:

- **Random Forest Evaluation:**
  - **Attack:** Excellent detection accuracy and precision for attacks.
  - **Benign:** High precision, indicating a few false positives; lower recall suggests some benign instances were misclassified.
  - **C&C:** Poor detection due to extremely low support and class imbalance.

- **C&C-HeartBeat:** High recall but low precision, which indicates this was detected in all instances but also had false positives.
- **C&C-Torii:** Detected some instances but missed many as observed from the low recall score.
- **DDoS:** Poor detection due to extremely low support and class imbalance.
- **Okiru:** Poor detection due to extremely low support and class imbalance.
- **PartOfAHorizontalPortScan:** Excellent recall and good overall detection, indicating reliability in identifying this type of attack.

- **Gradient Boosting Evaluation:**
  - **Attack:** Good precision and recall, indicating a balanced performance in detecting attacks.
  - **Benign:** High precision and recall, indicating strong performance in correctly identifying benign instances.
  - **C&C:** Poor detection due to very low support and class imbalance.
  - **C&C-HeartBeat:** Unable to detect this class due to low support.
  - **C&C-Torii:** Unable to detect this class due to low support.
  - **DDoS:** Some were detected but due to low recall, many DDoS instances were missed.
  - **Okiru:** Poor detection and low recall.
  - **PartOfAHorizontalPortScan:** Excellent recall and good precision, indicating reliable detection.
- **Ada Boost Evaluation:**
  - **Attack:** Poor detection, indicating an inability to identify attacks.
  - **Benign:** High precision and good recall, indicating strong performance in correctly identifying benign instances.
  - **C&C:** Very poor detection due to low support and class imbalance.
  - **C&C-HeartBeat:** Unable to detect this class, very low support.
  - **C&C-Torii:** Unable to detect this class, very low support.
  - **DDoS:** Unable to detect this class.
  - **Okiru:** Unable to detect this class.
  - **PartOfAHorizontalPortScan:** Excellent recall and good precision, indicating reliable detection.
- **Logistic Regression Evaluation:**
  - **Attack:** Excellent detection with near-perfect precision and recall.
  - **Benign:** High precision and good recall, indicating strong performance.
  - **C&C:** Unable to detect this class.
  - **C&C-HeartBeat:** Unable to detect this class.
  - **C&C-Torii:** High precision but very low recall, missing most instances.
  - **DDoS:** Unable to detect this class.
  - **Okiru:** Unable to detect this class.
  - **PartOfAHorizontalPortScan:** Excellent recall and good precision, indicating reliable detection.
- **Decision Tree Evaluation:**
  - **Attack:** Excellent detection with near-perfect precision and recall.
  - **Benign:** Excellent recall and good precision, indicating reliable detection.
  - **C&C:** Unable to detect this class.
  - **C&C-HeartBeat:** Perfect detection, though the support is very low, which might not be reliable or consistent when applied in real-world scenarios.

- **C&C-Torii:** High precision but low recall missing most instances, has very low support.
- **DDoS:** Unable to detect this class.
- **Okiru:** Poor detection.
- **PartOfAHorizontalPortScan:** Excellent recall and good precision, indicating reliable detection.

- **Light GBM Evaluation:**
  - **Attack:** Excellent detection with near-perfect precision and recall.
  - **Benign:** High precision and good recall, indicating strong performance.
  - **C&C:** High precision but very low recall and low support making detection unreliable.
  - **C&C-HeartBeat:** Perfect recall but very low precision, indicating it detected all instances but had false positives.
  - **C&C-Torii:** Low recall and moderate precision, missing most instances.
  - **DDoS:** Unable to detect this class.
  - **Okiru:** Poor detection.
  - **PartOfAHorizontalPortScan:** Excellent recall and good precision, indicating reliable detection.
- **Extra Trees Evaluation:**
  - **Attack:** Perfect detection with no false positives or false negatives.
  - **Benign:** High precision and good recall, indicating strong performance.
  - **C&C:** High precision but very low recall and low support making detection unreliable.
  - **C&C-HeartBeat:** Perfect recall, but low precision, indicating it detected all instances but also had false positives. Has very low support, which might not be reliable or consistent when applied in real-world scenarios.
  - **C&C-Torii:** Hight precision but low recall and low support, which might not be reliable or consistent when applied in real-world scenarios.
  - **DDoS:** Unable to detect this class.
  - **Okiru:** Poor detection.
  - **PartOfAHorizontalPortScan:** Excellent recall and good precision, indicating reliable detection.

4.5 Important Features from the Captured Network Packets

Several features were captured from network packets, each varying in importance when training different classifiers. Understanding the significance of these features is crucial in effectively implementing DT technology. By identifying the most important features, the accuracy and efficiency of malware detection and classification can be enhanced. This knowledge helps in focusing on collecting the right type of data, improving the current dataset, and addressing the heavily imbalanced type of malware being targeted.

Key features such as protocol types (proto_tcp, proto_udp), data volumes (orig_ip_bytes, resp_bytes), and connection states (conn_state) play a pivotal role in distinguishing between normal and malicious network activities. These features enable classifiers to better understand the behavior of different types of malwares and detect anomalies with a higher precision.

For instance, proto_tcp and proto_udp are critical in identifying the nature of the traffic, while orig_ip_bytes and resp_bytes provide insights into the data flow between originators and responders. Connection states, such as conn_state_s0 and conn_state_SF, help in recognizing incomplete or complete connections, which can be indicative of malicious activities.

By focusing on these important features, the model's ability to detect and classify malware efficiently can be enhanced, thus ensuring robust security measures. Additionally, prioritizing data collection based on their features allows for continuous improvement of the dataset, ultimately leading to a more accurate

and reliable malware detection system.

Figure 5 through Figure 10 provide a look at the feature importance of each category.

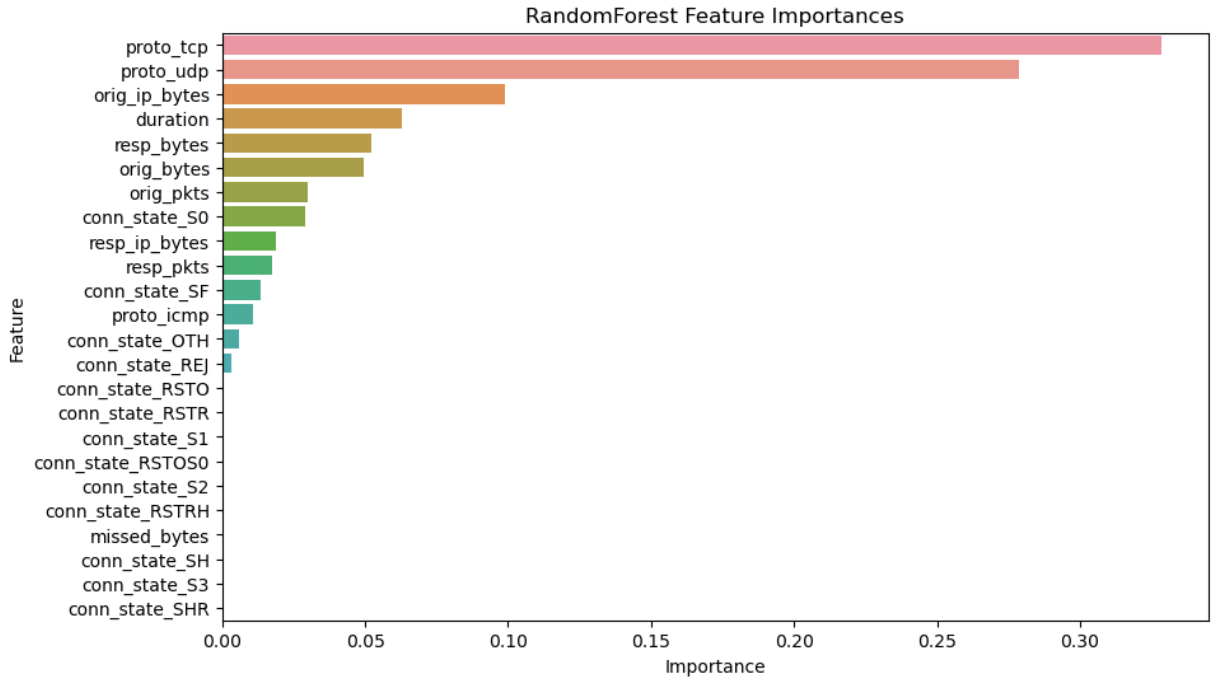

Figure 5. Random forest feature importance.

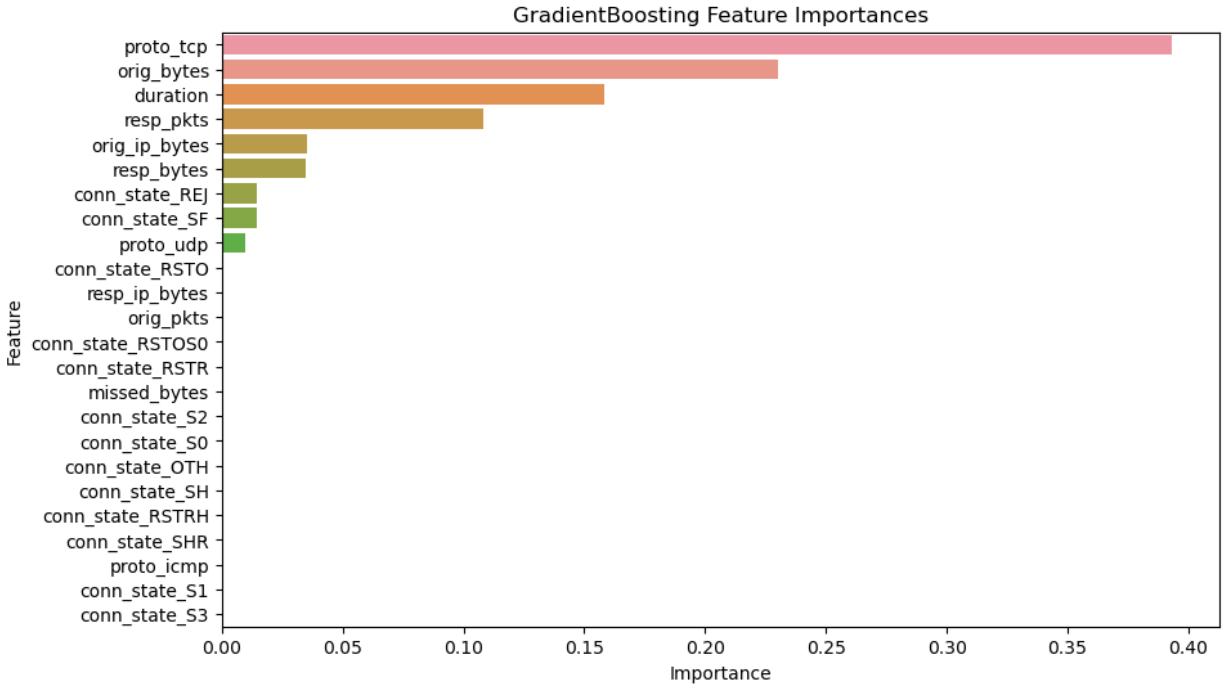

Figure 6. Gradient boosting feature importance.

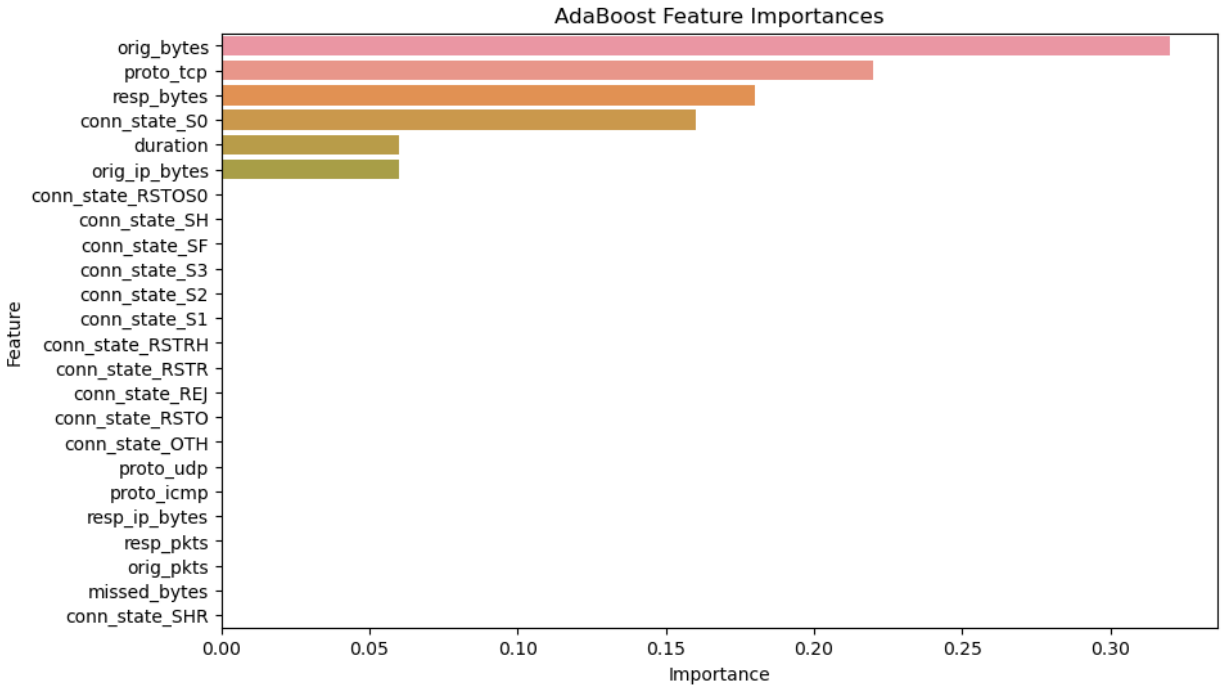

Figure 7. Ada boost feature importance.

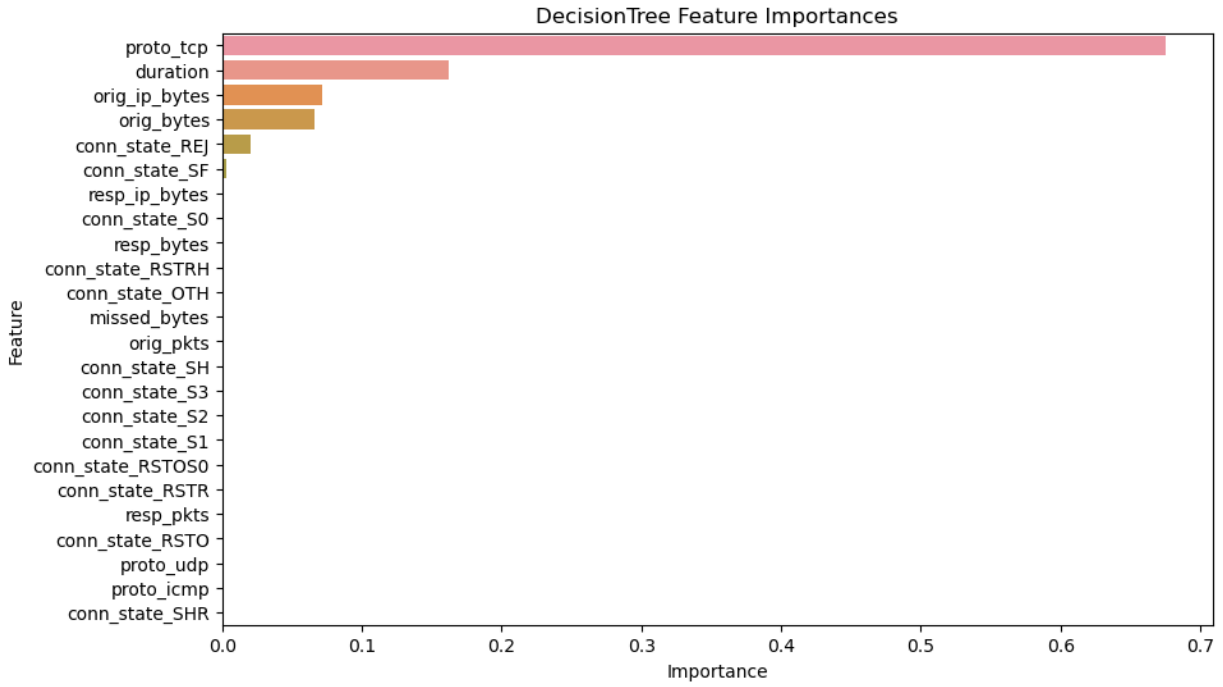

Figure 8. Decision tree feature importance.

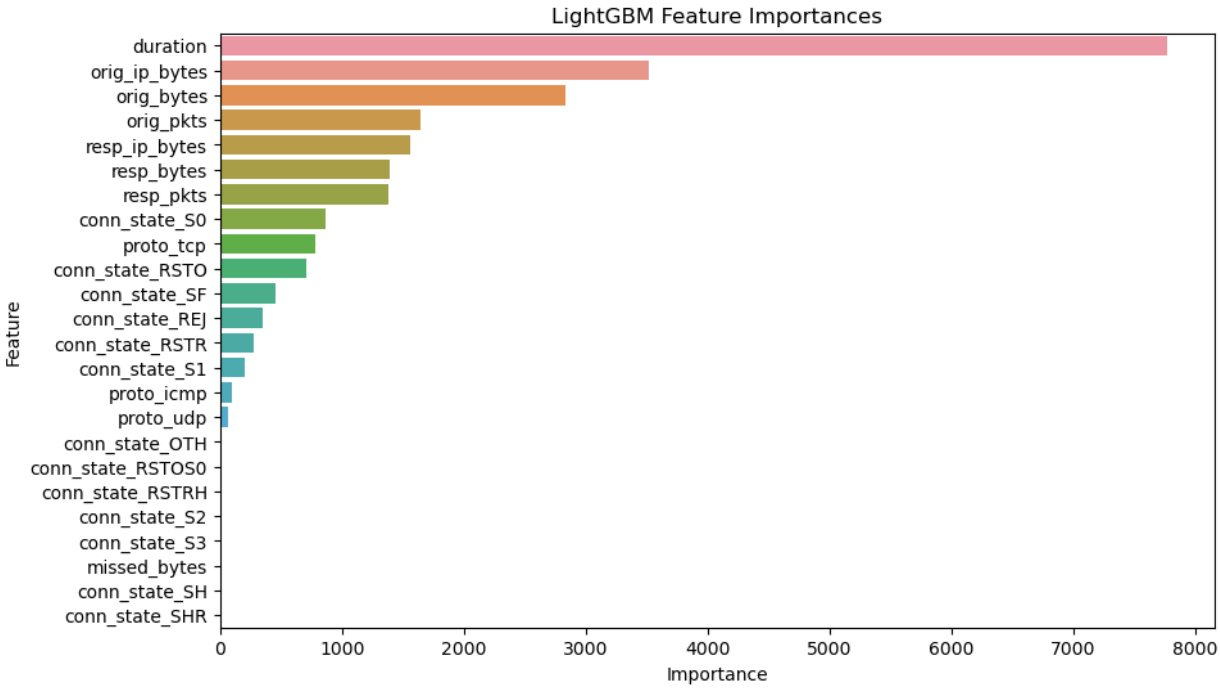

Figure 9. Light GBM feature importance.

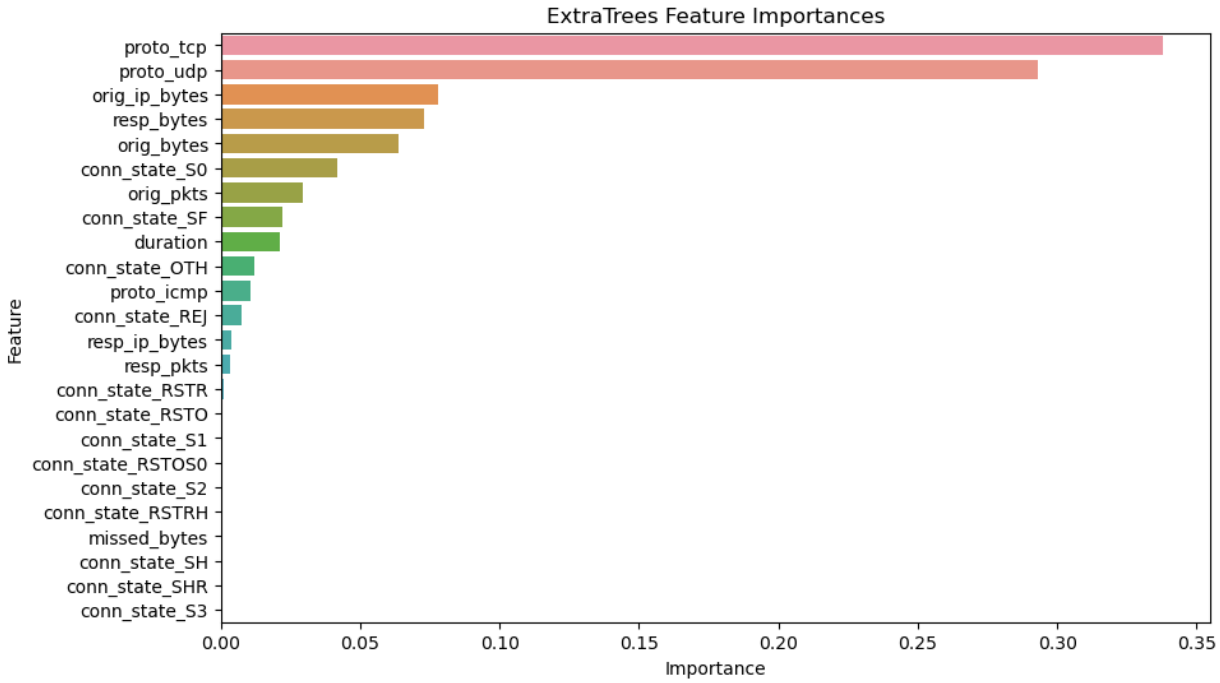

Figure 10. Extra trees feature importance.

5. MODEL SERIALIZATION ATTACKS

ML models are crucial assets in ML-powered applications, and their secure storage and retrieval are essential. Common model formats include Pickle, HDF5/H5, TensorFlow SavedModel, Model Checkpoints, and ONNX. These formats often store code alongside models, creating a potential security

risk. Models can be compromised through adversarial methods or traditional attacks. Model serialization attacks occur when malicious code is injected during the model saving and execution step. When a model is loaded for further training or inference, the attack code is executed immediately and often with no visible change in behavior. This makes model serialization attacks an easy point of entry for attacking broader ML components [33]. Various storage formats have different security implications:

1. **Pickle Variants:** Highly vulnerable to code injection attacks.
2. **TensorFlow SavedModel:** Generally secure, but certain operations, such as io.write_file and io.read_file, can still be exploited.
3. **H5(Keras):** Secure except for Lambda Layer operations, which allow arbitrary code execution.
4. **Inference Only:** Minimal risk, however, custom operations need careful evaluation.
5. **Vector/Tensor Only:** Secure except for Pickle variants. The potential exists for weight and bias manipulation, though this is not a simple task.

Except for Pickle variants, these formats cannot execute arbitrary code. However, an attacker can modify weights and biases to alter the ML model, thereby creating a security risk. Although it is challenging, meaningful manipulation of model weights and biases to perform a poisoning attack is possible.

5.1 Model Saving and Security

The scaled parameters and model wights were saved in .npy files for future use. The .npy file was chosen for serialization due to the security risk posed by other methods. Securing a trained model is crucial, as malware can inject malicious code into the model, thereby compromising the system where it runs. Table 11 outlines the risk level associated with each method [33].

Table 11. Model serialization methods risk level.

Approach	Popularity	Risk of Model Serialization Attack Exploitability
Pickle Variants	Very high	Very high
Tensorflow SavedModel	High	Medium
H5 (Keras)	High	Low (except Keras Lambda layer)
Inference Only	Medium	Low
Vector/Tensor Only	Low	Very low

5.1.1 How to Further Secure ML Models

Implementing defense-in-depth and zero trust strategies is essential for modern software security. The following measures have been recommended as well:

1. **Authenticated Access:** Only store ML models in systems with authenticated access.
2. **Authorization and Identify and Access Management (IAM):** Implement detailed least privilege access through authorization or IAM systems.
3. **Model Scanning:** Use a scanning tool like ModelScan to catch code injection attempts. Scan all models before use, whether retraining, fine-tuning, evaluation, or inference, at all points in an ML ecosystem.
4. **Encrypt Models at Rest:** Use a secure encryption method to reduce the chances of adversaries reading or writing models after a successful infiltration attempt.
5. **Encrypt models in Transit:** Always use secure connections to protect against Man-in-the-Middle (MITM) attacks when models are loaded over the network.

6. **Checksum Verification:** Store checksums and always verify them when loading models to ensure the integrity of the model files.
7. **Cryptographic Signatures:** Use cryptographic signatures to ensure model integrity and authenticity.

6. CONCLUSIONS

6.1 Summary of Key Findings

6.1.1 Class-wise Performance

All classifiers achieved a similar overall accuracy of around 80%. However, the performance varied significantly across different classes of malware:

- **Attack:** All classifiers except for Ada Boost showed excellent performance in detecting attack instances with near-perfect precision and recall.
- **Benign:** All classifiers demonstrated high precision and good recall for the benign class.
- **C&C:** Detection of this class was poor across all classifiers with very low recall and precision, likely due to the extreme low support for this class.
- **DDoS and Okiru:** These classes were detected poorly across all classifiers.
- **PartOfAHorizontalPortScan:** All classifiers performed well in detecting this class, with excellent recall and good precision, indicating reliable detection of horizontal port scans.

6.1.2 Feature Importances

The most important features across different classifiers were consistent, with “proto_tcp,” “proto_udp,” “orig_ip_bytes,” “duration,” “orig_bytes,” and “resp_bytes” being the frequent top features:

- **Random Forest:** “proto_tcp” and “proto_udp” were the top features indicating the significance of protocol-related features.
- **Gradient Boosting:** “proto_tcp” and “orig_bytes” were the top features.
- **Ada Boost:** “orig_bytes” was the top feature followed by “proto_tcp.”
- **Decision Tree:** “proto_tcp” and “duration” were the top features.
- **Light GBM:** “duration” followed by “orig_ip_bytes” were the top features.
- **Extra Trees:** “proto_tcp” and “proto_udp” were the top features indicating the significance of protocol-related features.

6.2 Future Research Directions

This project demonstrated the detection of malware in IoT devices and OT environments using network data. By analyzing network traffic with various ML models, malicious activities can be identified and characterized, thus enhancing the security in IoT environments. This approach provides a robust framework for real-time malware detection and characterization in IoT networks to identify infected devices and mitigate the necessary steps to remove or isolate the infected device to prevent further infections. With further enhancements, this model can be implemented within security systems of industrial networks, supervisory control and data acquisition (SCADA) systems, and other critical infrastructure environments. The deployment process would involve integrating the trained model into the network security infrastructure, continuously monitoring network traffic, and providing real-time alerts when anomalies or potential malware activities are detected. To further enhance the capability and efficiency of the malware and detection system, the following improvements are planned for future iterations:

1. **Create a Scalable Detection Framework:**

- Design and implement a scalable framework that can handle large volumes of network traffic data, providing real-time malware detection capabilities.
- Ensure the framework is adaptable to different types of IoT devices and network traffic data.

2. **Develop a Detailed Feature Selection Process:**
   - Implement advanced feature selection techniques to identify the most predictive attributes in the network traffic data.
   - Reduce data dimensionality to improve model effectiveness, ensuring faster and more accurate predictions.
3. **Standardize Data Collection and Preprocessing:**
   - Utilize the IoT-23 dataset collection and processing techniques to standardize and further collect data to add the existing dataset for better accuracy when training the classifiers.
   - Develop a comprehensive process for handling and preprocessing network traffic data including the cleaning, transformation, and integration of data from various sources.

7. REFERENCES

1. Fatani, A., M. Abd Elaziz, A. Dahou, M. A. A. Al-Qaness, and S. Lu. (2021). "IoT intrusion detection system using deep learning and enhanced transient search optimization." *IEEE Access*, 9, 123448–123464. https://doi.org/10.1109/ACCESS.2021.3109081.
2. Nie, L., Z. Ning, X. Wang, X. Hu, J. Cheng, and Y. Li. (2020). "Data-driven intrusion detection for intelligent Internet of vehicles: A deep convolutional neural network-based method." *IEEE Trans. Netw. Sci. Eng.*, 7(4), 2219–2230. https://doi.org/10.1109/TNSE.2020.2990984.
3. Ullah, I., and Q. H. Mahmoud. (2021). "Design and development of a deep learning-based model for anomaly detection in IoT networks." *IEEE Access*, 9, 103906–103926. https://doi.org/10.1109/ACCESS.2021.3094024.
4. Liang, W., Y. Hu, X. Zhou, Y. Pan, and K. I.-K. Wang. (2021). "Variational few-shot learning for microservice-oriented intrusion detection in distributed industrial IoT." *IEEE Trans. Ind. Inf.*, 18(8), 5087–5095. https://doi.org/10.1109/TII.2021.3116085.
5. Muthanna, M. S. A., R. Alkanhel, A. Muthanna, A. Rafiq, and W. A. M. Abdullah. (2022). "Towards SDN-enabled, intelligent intrusion detection system for Internet of Things (IoT)." *IEEE Access*, 10, 22756–22768. https://doi.org/10.1109/ACCESS.2022.3153716.
6. Zeeshan, M., Q. Riaz, M. A. Bilal, M. K. Shahzad, H. Jabeen, S. A. Haider, and A. Rahim. (2021). "Protocol-based deep intrusion detection for DoS and DDoS attacks using UNSW-NB15 and Bot-IoT data-sets." *IEEE Access*, 10, 2269–2283. https://doi.org/10.1109/ACCESS.2021.3137201.
7. Qiu, H., T. Dong, T. Zhang, J. Lu, G. Memmi, and M. Qiu. (2020). "Adversarial attacks against network intrusion detection in IoT systems." *IEEE Internet Things J.*, 8(13), 10327–10335. https://doi.org/10.1109/JIOT.2020.3048038.
8. Ibitoye, O., O. Shafiq, and A. Matrawy. (2019). "Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks." *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6. https://doi.org/10.1109/GLOBECOM38437.2019.9014337.
9. Mehedi, S. T., A. Anwar, Z. Rahman, K. Ahmed, and R. Islam. (2022). "Dependable intrusion detection system for IoT: A deep transfer learning-based approach." *IEEE Trans. Ind. Inf.*, 19(1), 1006–1017. https://doi.org/10.1109/TII.2022.3164770.
10. Zhou, X., W. Liang, W. Li, K. Yan, S. Shimizu, and K. I.-K. Wang. (2021). "Hierarchical adversarial attacks against graph-neural-network-based IoT network intrusion detection system." *IEEE Internet Things J.*, 9(12), 9310–9319. https://doi.org/10.1109/JIOT.2021.3130434.

11. Wahab, O. A. (2022) “Intrusion detection in the IoT under data and concept drifts: Online deep learning approach.” *IEEE Internet Things J.*, 9(20), 19706–19716. https://doi.org/10.1109/JIOT.2022.3167005.
12. Alkadi, O., N. Moustafa, B. Turnbull, and K.-K. R. Choo. (2020). “A deep blockchain framework-enabled collaborative intrusion detection for protecting IoT and cloud networks.” *IEEE Internet Things J.*, 8(12), 9463–9472. https://doi.org/10.1109/JIOT.2020.2996590.
13. Taher, F., M. Elhoseny, M. K. Hassan, and I. M. El-Hasnony. (2022). “A novel tunicate swarm algorithm with hybrid deep learning enabled attack detection for secure IoT environment.” *IEEE Access*, 10, 127192–127204. https://doi.org/10.1109/ACCESS.2022.3226879.
14. Abdel-Basset, M., H. Hawash, R. K. Chakrabortty, and M. J. Ryan. (2021). “Semi-supervised spatiotemporal deep learning for intrusions detection in IoT networks.” *IEEE Internet Things J.*, 8(15), 12251–12265. https://doi.org/10.1109/JIOT.2021.3060878.
15. Alani, M. M., and A. I. Awad. (2022). “An intelligent two-layer intrusion detection system for the Internet of Things.” *IEEE Trans. Ind. Inf.*, 19(1), 683–692. https://doi.org/10.1109/TII.2022.3192035.
16. Thamilarasu, G., and S. Chawla. (2019). “Towards deep-learning-driven intrusion detection for the Internet of Things.” *Sensors* 19(9), 1977. https://doi.org/10.3390/s19091977.
17. Zhang, Y., P. Li, and X. Wang. (2019). “Intrusion detection for IoT based on improved genetic algorithm and deep belief network.” *IEEE Access*, 7, 31711–31722. https://doi.org/10.1109/ACCESS.2019.2903723.
18. de Elias, E. M., V. S. Carriel, G.W. De Oliveira, A. L. Dos Santos, M. Nogueira, R. H. Junior, and D. M. Batista. (2022). “A hybrid CNN-LSTM model for IIoT edge privacy-aware intrusion detection.” *2022 IEEE Latin-American Conference on Communications (LATINCOM)*, Rio de Janeiro, Brazil, pp. 1–6. https://doi.org/10.1109/LATINCOM56090.2022.10000468.
19. Khacha, A., R. Saadouni, Y. Harbi, and Z. Aliouat. (2022). “Hybrid deep learning-based intrusion detection system for Industrial Internet of Things.” *2022 5th International Symposium on Informatics and its Applications (ISIA)*, M’sila, Algeria, pp. 1–6. https://doi.org/10.1109/ISIA55826.2022.9993487.
20. Jahromi, A. N., H. Karimipour, and A. Dehghantanha. (2023). “An ensemble deep federated learning cyber-threat hunting model for Industrial Internet of Things.” *Comput. Commun.*, 198, 108–116. https://doi.org/10.1016/j.comcom.2022.11.009.
21. Ahmad, R., I. Alsmadi, W. Alhamdani, and L. Tawalbeh. (2022). “A deep learning ensemble approach to detecting unknown network attacks.” *J. Inf. Secur. Appl.*, 67, 103196. https://doi.org/10.1016/j.jisa.2022.103196.
22. Al-Hamadi, H., I.-R. Chen, D.-C. Wang, and M. Almashan. (2020). “Attack and defense strategies for intrusion detection in autonomous distributed IoT systems.” *IEEE Access*, 8, 168994–169009. https://doi.org/10.1109/ACCESS.2020.3023616.
23. Eskandari, M., Z. H. Janjua, M. Vecchio, and F. Antonelli. (2020). , Passban IDS: An intelligent anomaly-based intrusion detection system for IoT edge devices.” *IEEE Internet Things J.*, 7(8), 6882–6897. https://doi.org/10.1109/JIOT.2020.2970501.
24. Abu Al-Haija, Q., S. Zein-Sabatto. (2020). “An efficient deep-learning-based detection and classification system for cyber-attacks in IoT communication networks.” *Electronics*, 9(12), 2152. https://doi.org/10.3390/electronics9122152.
25. Gad, A. R., A. A. Nashat, and T. M. Barka. (2021). “Intrusion detection system using machine learning for vehicular ad hoc networks based on ToN-IoT dataset.” *IEEE Access*, 9, 142206–142217. https://doi.org/10.1109/ACCESS.2021.3120626.

26. Shafiq, M., Z. Tian, A. K. Bashir, X. Du, and M. Guizani. (2020). "CorrAUC: A malicious bot-IoT traffic detection method in IoT network using machine-learning techniques." *IEEE Internet Things J.*, 8(5), 3242–3254. https://doi.org/10.1109/JIOT.2020.3002255.
27. Gao, Y., J. Chen, H. Miao, B. Song, Y. Lu, and W. Pan. (2022). "Self-learning spatial distribution-based intrusion detection for industrial cyber-physical systems." *IEEE Trans. Comput. Soc. Syst.*, 9(6), 1693–1702. https://doi.org/10.1109/TCSS.2021.3135586.
28. Nie, L., W. Sun, S. Wang, Z. Ning, J. J. P. C. Rodrigues, Y. Wu, and S. Li. (2021). "Intrusion detection in green internet of things: A deep deterministic policy gradient-based algorithm." *IEEE Trans. Green Commun. Netw.*, 5(2), 778–788. https://doi.org/10.1109/TGCN.2021.3073714.
29. Rezaei, T., F. Manavi, and A. Hamzeh. (2021). "A PE header-based method for malware detection using clustering and deep embedding techniques." *J. Inf. Secur. Appl.*, 60, 102876. https://doi.org/10.1016/j.jisa.2021.102876.
30. Chebbi, C. (2018). Mastering machine learning for penetration testing: Develop an extensive skill set to break self-learning systems using Python. Packt Publishing, Ltd., Birmingham, U.K. Available at: https://digtvbg.com/files/books-for-hacking/Mastering%20Machine%20Learning%20for%20Penetration%20Testing%20-%20Develop%20an%20extensive%20skill%20set%20to%20break%20self-learning%20systems%20using%20Python%20by%20Chiheb%20Chebbi.pdf (accessed 4 September 2024).
31. GitHub.com. (2021). "Anomaly Detection IoT23." User: yliang725. Available at: https://github.com/yliang725/Anomaly-Detection-IoT23 (accessed 15 August 2024).
32. Garcia, S., A. Parmisano, and M. J. Erquiaga. (2020). "IoT-23: A labeled dataset with malicious and benign IoT network traffic (Version 1.0.0) [Data set]." Zenodo website. http://doi.org/10.5281/zenodo.4743746.
33. GitHub.com. (2024). "ModelScan: Protection Against Model Serialization Attacks." User: protectai. Available at: https://github.com/protectai/modelscan (accessed 15 August 2024).