Detecting IoT Botnets on IoT Edge Devices

Meghana Raghavendra
Computer Science
Purdue University Fort Wayne
Fort Wayne, USA
raghm01@pfw.edu

Zesheng Chen
Computer Science
Purdue University Fort Wayne
Fort Wayne, USA
chenz@pfw.edu

2022 IEEE International Conference on Communications Workshops (ICC Workshops) | 978-1-6654-2671-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCWORKSHOPS53468.2022.9814555

Abstract— **Rapid expansion in the utilization of Internet of things (IoT) devices in everyday life leads to an increase in the attack surface for cybercriminals. IoT devices have been frequently compromised and used for the creation of botnets. The goal of this research is to identify a machine learning method that can be run on resource-constrained IoT edge devices to detect IoT botnet traffic accurately in real time. Specifically, we apply both the input perturbation ranking (IPR) algorithm and decision trees to achieve this goal. We study the network snapshots of IoT traffic infected with two botnets, *i.e.*, Mirai and Bashlite, and use IPR with XGBoost to identify nine most important features that distinguish between benign and anomalous traffic for IoT devices. We propose to use decision trees, a supervised machine learning method, because of its simplicity, less time to train and predict, ease to be translated to security policy, and flexibility on balancing detection accuracy and speed. In our experiments, we compare the performance of decision trees with a deep-learning based method, *i.e.*, Kitsune, and other popular supervised machine learning methods. We show that decision trees are with high decision performance (*e.g.*, more than 99.99% accuracy), but with much less training and prediction time than Kitsune and most other machine learning methods. Moreover, we demonstrate that using nine most important features in decision tress, the detection accuracy is similar, but the computation power can be significantly reduced, making botnet detection suitable on IoT edge devices.**

Keywords— Internet of Things, Botnets, Decision Trees, Input Perturbation Ranking Algorithm.

I. Introduction

The Internet of things (IoT) describes a network of physical objects that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet [1]. These IoT devices, ranging from security cameras to baby monitors, are constantly streaming private data throughout the Internet but without strong security protections. This leads to a large number of devices that can be easily compromised and used as bots in IoT botnets [2]. IoT botnets are one of the most common attacks against the IoT devices. Attackers compromise less secure IoT devices and convert them into a network of bots, which are used to conduct distributed denial of service (DDoS) attacks on targeted devices, systems, or websites [2]. Therefore, it is important and imperative to detect the appearance of IoT botnets accurately and in real time.

Traditionally, a powerful machine is used to run a network intrusion detection system (NIDS) in an IoT network. However, the NIDS can potentially stop functioning properly due to many reasons such as misconfiguration or being under DDoS attacks. As a result, it is desirable that an IoT edge device can have a certain capability to detect botnet traffic accurately and quickly.

Different machine learning algorithms have been explored to detect IoT botnet traffic in the network in real time [3], [4], [5], [6]. One of the most popular methods used is an unsupervised deep learning method called autoencoders. An autoencoder is a type of artificial neural network used to learn efficient data representation in an unsupervised manner, where the output is the same as the input during training to obtain the characteristics of the input data [3], [5]. Specifically, Kitsune [3], the state-of-the-art detector against IoT botnets, uses the features of benign traffic as both input and output to the autoencoders during training. The advantage of using autoencoders is that there is no need for a previously labelled dataset during the training period. While this is a huge advantage to detect network attacks, the time and memory requirements to run autoencoders such as Kitsune are still significant and cannot fit on resource-constrained IoT edge devices such as the Raspberry Pi [7]. As a result, it is still a research question: what is a proper machine learning method that can be run on resource-constrained IoT edge devices to detect botnet traffic accurately and in real time, in order to form the first line of defense in an IoT network?

To answer this question, we begin with studying two botnets – Mirai and Bashlite. Specifically, the Mirai botnet is a collection of IoT bots that attacked several high-profile targets with massive DDoS attacks [8]. The Mirai botnet came into mainstream during 2016 mainly due to the increased usage of IoT devices that had insecure default passwords configured and lacked embedded security measures. Bashlite or Gafgyt is a well-known malware that infects Linux-based IoT devices to create bots to launch DDoS attacks [9]. It converted Linux-based IoT devices into bots by brute forcing their default Telnet access. We obtained the datasets containing benign and anomalous Mirai and Bashlite network traffic for nine different IoT devices from UC Irvine machine learning repository [10]. Each dataset consists of 115 features extracted from network traffic.

Next, we attempt to identify the most important features among 115 features, by using the input perturbation ranking (IPR) algorithm [11], [12]. Specifically, we applied the IPR method with the XGBoost machine learning algorithm [13], [14], which is a powerful tool in machine learning, to each of the nine datasets (*i.e.*, nine IoT devices). The IPR method is able to rank input features based on their effects on the predictions of XGBoost in a dataset. By studying ranking results from all datasets, we are able to find nine most important features that can essentially be used to differentiate between benign and anomalous traffic. As shown in this paper, when these nine most important features are applied, the processing time for both training and prediction is significantly less than that of the original 115 features. Moreover, the detection accuracy is over 99.99%, which is still very high.

In this work, we propose to use the decision trees machine learning method [13] to detect IoT botnet attacks in real time and to run on resource-constrained IoT edge devices (*i.e.*, with low computation power and low memory) as the first line of defense. Specifically, we use a weighted decision-tree classifier with varying maximum depths for the detection of IoT botnets on different IoT edge devices. A decision tree is a predictive supervised machine learning algorithm that utilizes the input features of a packet to predict if the packet is benign or anomalous [13]. The advantages of using decision trees over other machine learning methods include:

- **Fast detection and low resource requirements –** It is well known that decision trees require low computation power and are fast to train and predict [13]. Moreover, comparing with other machine learning methods, decision trees demand less memory to store model information. Hence, such a machine learning method can be easily implemented on resource-constrained IoT edge devices.
- **Security policy making -** Decision trees can be used to visually and explicitly represent decisions and decision-making processes. In a decision tree, decision nodes represent the clear conditions to distinguish between benign and anomalous traffic, which can be easily translated into explicit security policies.
- **Tradeoff between accuracy and computation power by varying maximum tree depths –** Decision trees provide the user the flexibility to choose between accuracy and computational power. The user can select to have a low maximum tree depth with slightly less accuracy and a higher false positive rate, but with significantly lower computational power. The taller the tree is allowed to grow, the higher the accuracy. This flexibility is not seen with other machine learning methods such as autoencoders, where the user is unable to control how big the autoencoder can grow or how much computation power will be used.

From our experiments, we have found that the decision tree is able to match (and sometimes is greater than) the accuracy of Kitsune and other popular machine learning such as XGBoost, while taking much less time for both training and prediction. For example, as shown in Table III, decision trees run more than 25 times faster for training and 126,553 times faster for prediction than Kitsune on a Raspberry Pi 4 with 8 GB RAM.

The rest of the paper is divided into the following sections: Section II introduces the datasets used. Sections III and IV describe the process of identifying most important features and applying decision trees, respectively. Section V provides the experiment results of our algorithms and compares the performance of decision trees with that of other widely used machine learning methods. Finally, Section VI presents our conclusions and discusses the future work.

II. Dataset Description

The datasets used in our work consist of 115 features extracted from network traffic flowing through nine different IoT devices [10] such as webcam, doorbells, thermostat, security cameras, and baby monitor (see Table I). The datasets contain both benign traffic and anomalous Mirai and Bashlite traffic.

Table I: *Description of the nine datasets*

Dataset Name	Benign Samples	Malicious Samples	Mirai Botnet Data	Bashlite Botnet Data
Danmini Doorbell	49548	968750	✓	✓
Ecobee Thermostat	13113	822763	✓	✓
Ennio Doorbell	39100	316400	X	✓
Philips B120N10 Baby Monitor	175240	923437	✓	✓
Provision PT737E Security Camera	62154	766106	✓	✓
Provision PT838 Security Camera	98514	738377	✓	✓
Samsung SNH1011N Webcam	52150	323072	X	✓
SimpleHome XCS71002WHT Security Camera	46585	816471	✓	✓
SimpleHome XCS71003WHT Security Camera	19528	831298	✓	✓

Each IoT device has its own functionality and a different network traffic pattern that is extracted into 23 features from 5 different time windows of the most recent 100 ms, 500 ms, 1.5 sec, 10 sec, and 1 min [5]. The features include packet size, packet count, and packet jitter based on host IP address, host MAC address, channel, and socket. Moreover, aggregation methods [10], such as mean, variance, and correlation, have been used to obtain these features of network traffic.

In the dataset, each feature is named using the format of "Source_Type_LTimeWindow_Aggregation". For example, feature "HH_jit_L5_mean" refers to the average of the packet jitter between the packet's source and destination over a particular channel in the time window of 1 min. Please refer to [5] for the detailed information of the datasets.

III. Important Feature Ranking

In machine learning, finding important features is essential for making accurate predictions. For the datasets, we attempt to determine the most important features to differentiate between normal and anomalous traffic by using the input perturbation ranking (IPR) method [12] with the XGBoost algorithm [13].

IPR is a feature importance algorithm that calculates the loss of a model when each of the input features to the model is perturbed by the algorithm. When a feature is perturbed, it becomes useless and is equivalent to being removed from the evaluation. Thus, if a feature is more important, such perturbation will lead to more losses and less accurate prediction results for the model. In IPR, the XGBoost algorithm is applied, as it leverages the advantages of random forests and gradient boosting to strengthen the model and provides prediction errors ten times lower than the random forests method, but with significantly lower runtime [14]. Once the XGBoost model is trained, we perturb each input feature in the test dataset and then predict the target probabilities. By running the IPR, we are able to collect 30 most important features for each of nine datasets based on the prediction losses from the 115 features and rank these features from one to 30 with one being the most important and 30 being the least important. Next, we filter these nine features lists by identifying the common features that appear across all nine lists. There are only nine features shown in all nine lists. For each of these nine features, we then add the rank numbers across the nine datasets and get a final rank score. For example, if feature A is ranked number one in dataset one, number four in dataset two, and so on, we add the rank numbers to get an aggregated rank score. The smaller the rank score, the more important the feature is. Finally, we ended up retrieving nine most important features with their rank scores as shown in Table II.

Table II: *Nine most important features*

Features	**Rank Score**
HH_jit_L5_mean	**14**
MI_dir_L0.1_weight	**31**
HH_jit_L0.01_mean	**35**
MI_dir_L0.01_mean	**76**
HH_jit_L0.01_variance	**87**
HH_L0.01_radius	**91**
MI_dir_L0.01_weight	**91**
HH_jit_L1_variance	**130**
MI_dir_L0.01_variance	**134**

As described in Section V, when these nine important features are applied to the decision trees classifier running on a Raspberry Pi 4, the detection accuracy is over 99.99% for all datasets, and the processing time is in the form of a few seconds. This shows that these nine features can be used as main features to detect any IoT malicious traffic with very high accuracy and very small processing time. Moreover, it helps us in understanding network features that might be a common factor in differentiating between normal and anomalous IoT traffic. For example, "HH_jit_L5_mean" is the most important feature in the list. When looking into the values of this feature in the datasets, we found that comparing with normal traffic, the jitter in anomalous traffic tends to be either very high or extremely low, which may characterize a pattern in IoT botnets.

IV. Decision Tree Classifier

In this work, we propose to use decision trees to detect botnet traffic in different types of IoT devices. Decision trees require low computational and memory resources for detection and thus can be applied as a first line of defense on resource-constrained IoT edge devices. Moreover, the decision tree model provides clear and specific rules, based on which the model classifies the samples as normal or anomalous. These clear and concise rules can be easily converted into security policies and applied on network firewalls to prevent malicious traffic from infiltrating the network. Another advantage that decision trees provide over other machine learning methods such as autoencoders or XGBoost is that we can have a tradeoff between computational resources required and detection accuracy, by varying the maximum depth of a decision tree (*i.e.*, "max_depth" parameter). Specifically, in the case of IoT devices with very low computational resources such as doorbells or thermostats, running decision trees with a small max_depth can help with detection, but without greatly affecting its functionality. On the other hand, in devices with more resources such as security cameras, the max_depth can be increased proportionally to ensure more efficient detection performance. Hence, the user is given the flexibility to balance between the detection performance and the resources of an IoT device.

The decision trees method is a predictive supervised machine learning algorithm. It can predict the class of the target variable by learning simple rules from a prior training dataset. The crux of the decision tree split depends on choosing the input features that best differentiate between classes. In our method, we apply the Gini Index as a measure for splitting the decision tree nodes. Specifically, the Gini index [13] is used to determine the important features based on which the tree will be split into sub-nodes. The Gini index measures the probability of a sample being classified in the wrong class in the following equation.

$$Gini\ Index = 1 - \sum_{i=1}^{n} {p_i}^2$$

where n is the number of classes and p_i is the probability of a sample being classified to a particular class. The Gini index varies between *0* and $1 - 1/n$, with *0* showing that all samples belong to the class that they were classified into and $1 - 1/n$ showing that the samples are evenly distributed between the classes. Hence, a lower value of the Gini index in the leaves of the tree implies greater discrimination.

Since the datasets contain a larger number of anomalous traffic as compared to normal traffic, weighted decision trees are applied to train the model. By using class weights that are inversely proportional to the sizes of class data samples, the decision tree becomes balanced and impartial.

The max_depth of the decision tree is an important consideration in our work. If the max_depth is undefined (*i.e.*, "none"), the decision tree is allowed to grow to its full length, until no more nodes can be split into sub-nodes. On the other hand, if the max_depth is defined and becomes smaller, it will require fewer resources and take a shorter time to train the decision trees model, but have a lower detection accuracy, which indicates the tradeoff between detection performance and computational resources required. A preliminary experiment for the values of max_depth shows that a value of 3 has results that are comparable to those of the undefined max_depth. Hence, we limit our discussions to using 1, 2, 3, and none for max_depth.

V. Performance Evaluation

In this section, we evaluate the performance of our method in terms of detection and runtime performance. We first describe the experimental setup and then compare our method with Kitsune. Next, we further compare the performance of decision trees with that of seven supervised machine learning methods. Finally, we show the experimental results when the nine important features are applied to decision trees.

It is noted that Kitsune is an unsupervised online learning method, whereas the decision trees method is a supervised machine learning algorithm. We compared them in an effort to show that they have the similar detection accuracy, but with differences in runtime performance. Specifically, decision trees require significantly less runtime than Kitsune. This indicates that using a simpler method as a first line of defense is more efficient than using a computation heavy method, especially for the case of resource-constrained IoT edge devices.

A. *Experimental Setup*

We divided each of the nine datasets into training and test sets with an 80-20 distribution. The training dataset is used to train a model, whereas the test dataset is applied to evaluate the performance of the model. The same set of training and test datasets is applied to all different machine learning methods. Since Kitsune is an unsupervised machine learning method, only benign samples in the training dataset are used to train Kitsune. Table I describes the sample counts of benign and malicious traffic for each of the datasets used in the experiments. Moreover, benign traffic is assigned as class 0, and anomalous traffic is assigned as class 1. To measure the detection performance of a model, we use performance metrics such as accuracy, precision, recall, F1-score, true positive rate (TPR), and false positive rate (FPR) [13].

B. *Comparisons Between Decision Trees and Kitsune*

To fairly compare decision trees with the baseline, *i.e.*, Kitsune, here we apply all 115 features, instead of nine most important features, in decision trees. The experiments were conducted on a Raspberry Pi 4 with 8 GB RAM and a 1.5GHz Quad-core ARM v8 processor. In Table III, the performance of a decision tree with the undefined max_depth is compared with that of the Kitsune autoencoder baseline for all nine datasets. It can be seen that the runtime of decision trees is significantly lower. The average training time for decisions trees is about 2.7 minutes, and the average prediction time is only 0.18 seconds. In comparison, Kitsune requires about 71.4 minutes and 6.3 hours for training and prediction, respectively. Moeover, it can be seen that the detection performance of decision trees is similar or better than that of Kitsune. Specifically, decision trees have an average accuracy of 99.997%, which is slightly better than that of Kitsune (*i.e.*, 99.496%). Furthermore, the average FPR of Kitsune is 6.697%, whereas that of decision trees is only 0.019%. For Kitsune, all TPR is close to 100%, but FPR is in the range of 3.699% to 10.222%. This shows that while Kitsune is extremely good at detecting the anomalous traffic patterns, it falters a bit by raising a higher number of false alarms. On the other hand, the FPR of decision trees is in the range of 0% to 0.038%, which indicates that the decision trees algorithm is much better at detecting normal traffic. Through comparison, it can be clearly seen that our proposed method has a better performance as an IoT botnet detector, especially on resource-constrained IoT edge devices with low computation power.

Figure 1 shows the confusion matrix of decision trees with different max_depth for the Danmini Doorbell dataset. It can be seen that when the max_depth increases, the detection performance becomes better. On the other hand, the training time is 22.874, 42.82, 69.943, and 196.781 seconds for decision trees with the max_depth of 1, 2, 3, and none, which indicates the tradeoff between accuracy and runtime performance.

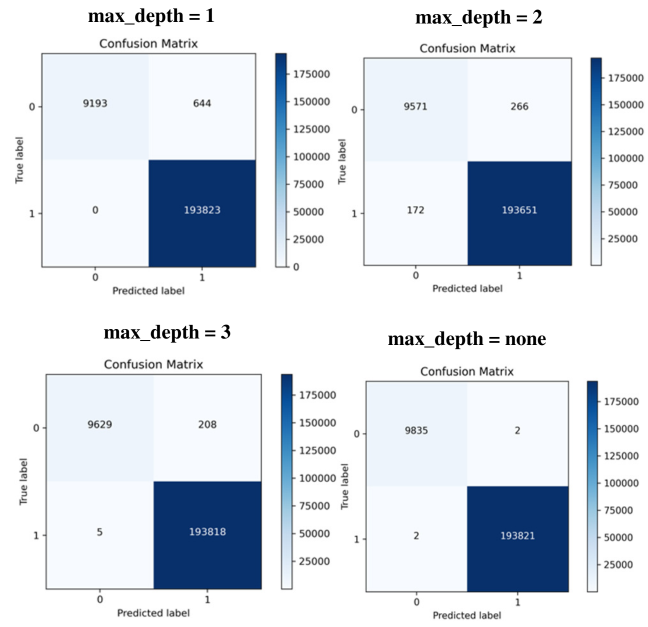

Figure 1*: Confusion matrix of decision trees with different max_depth for the Danmini Doorbell dataset*

Table III*: Comparing the performance of decision trees (with the undefined max_depth) with that of Kitsune Baseline*

Dataset	Decision Trees						Kitsune - Autoencoders					
	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	F1-Score (%)	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	F1-Score (%)
Danmini Doorbell	196.781	0.239	99.999	0.02	99.998	99.999	6173.176	34481.68	100	3.873	99.813	99.902
Ecobee Thermostat	161.973	0.185	99.998	0.038	99.997	99.998	2049.148	22109.539	100	9.923	99.845	99.921
Ennio Doorbell	63.326	0.085	99.998	0.038	99.994	99.997	6652.826	13233.602	99.992	3.699	99.584	99.766
Philips B120N10 Baby Monitor	291.377	0.241	99.998	0.011	99.996	99.998	5341.764	46076.255	99.995	5.733	99.088	99.461
Provision PT737E Security Camera	169.813	0.1866	99.997	0.032	99.995	99.997	4584.981	21924.251	99.999	5.561	99.578	99.772
Provision PT838 Security Camera	187.286	0.201	99.999	0.005	99.998	99.999	3836.054	21683.915	99.999	6.653	99.216	99.557
Samsung SNH1011N Webcam	56.221	0.086	99.995	0	99.996	99.998	4044.547	9468.293	99.997	6.436	99.095	99.477
SimpleHome XCS71002WHT Security Camera	127.539	0.198	99.999	0	99.999	99.999	3837.574	18107.613	99.994	10.222	99.435	99.702
SimpleHome XCS71003WHT Security Camera	202.073	0.194	99.999	0.026	99.998	99.999	2010.712	17933.029	100	8.173	99.813	99.904
AVERAGE	**161.821**	**0.18**	**99.998**	**0.019**	**99.997**	**99.998**	**4281.198**	**22779.797**	**99.997**	**6.697**	**99.496**	**99.718**

Table IV: *Comparing the performance of decision trees with that of other supervised machine learning methods*

ML Algorithms	**Training Time (seconds)**	**Prediction Time (seconds)**	**TPR (%)**	**FPR (%)**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1-Score (%)**
Decision Tree (115 Features)	1024.186	4.971	100	0.001	100	100	100	100
Decision Tree (9 Features)	46.93	0.078	100	0.004	100	100	100	100
Logistic Regression	4755.496	4.304	99.958	0.167	99.948	99.986	99.958	99.972
Naïve Bayes	167.702	7.175	99.953	40.735	96.761	96.648	99.953	98.272
Random Forests	2880.949	13.104	100	0.001	100	100	100	100
AdaBoost	13868.753	61.984	100	0.005	100	100	100	100
XGBoost	3583.153	5.701	100	0.001	100	100	100	100
SGD	37.815	3.787	99.91	0.205	99.901	99.983	99.91	99.946
ANN	17382.242	20.619	99.999	0.084	99.993	99.993	99.999	99.996

C. *Combined Dataset Classification*

It is feasible to combine the datasets together to form one dataset and then run it through the decision trees classifier. However, this is not possible using Kitsune autoencoders, as combining the datasets will remove the patterns of specific devices and hence make the autoencoder models unusable. In this experiment, we mixed the nine datasets into one single dataset with 115 input features and run it through a binary decision trees classifier with the undefined max_depth, as well as seven other widely used machine learning methods. Specifically, we consider logistic regression, naïve Bayes, random forests, AdaBoost, XGBoost, stochastic gradient descent (SGD), and artificial neural network (ANN) classifiers [13]. For these seven methods, the 115 input features in the dataset were normalized using the min-max normalization technique [13] with all the values fitting between 0 and 1. The experiment was run on a MacBook Pro with 8GB RAM and a 2.4GHz Quad-core Intel Core i5 processor. Table IV shows the detection performance, training time, and prediction time of seven popular machine learning methods compared with our method. It can be seen that malicious botnet traffic across different IoT devices can be detected accurately as a whole. Decision trees, random forests, AdaBoost, and XGBoost all have a detection accuracy of 100%. Among these four machine learning methods, decision trees using 115 features took much less time to train and predict. Specifically, the time for decision trees (with 115 features) to train with 5,650,084 samples was 1024.186 seconds, and the time to predict 1,412,522 samples was only 4.971 seconds. Although logistic regression and SGD classifiers took less time for prediction, the detection accuracy of these two detectors was not as good as that of decision trees. The SGD classifier used less training time than decision trees due to the fact that the SGD classifier is an optimized method that runs on the top of linear support vector machine (SVM) classifier [13]. Although the SGD classifier is very fast, its accuracy is lower than that of the decision tree method. Moreover, SGD cannot provide the explicit tradeoff between detection accuracy and computational resources required, which detection trees are able to provide.

Here we did not show the comparison with the k-nearest neighbors (KNN) algorithm [13], as it is well known that KNN is an extremely slow learning method that calculates the distance (*e.g.*, Euclidean) of each of predicted samples with its k-nearest neighbors. This would take an ample amount of runtime for 1 million test samples. As a result, KNN is not considered here.

In our experiment, an ANN model is built with two hidden layers using the rectified linear unit (ReLU) activation function and the output layer using the sigmoid activation function for binary classification. The model is configured with the Adam function for the optimization method and the binary cross-entropy function for loss measurement. The model is trained for 150 epochs. From Table IV, it can be seen that the ANN model has a high accuracy of 99.993% with training time being more than four hours and the prediction time being comparatively low, *i.e.*, about 20 seconds. Compared to our proposed method, ANN took much longer runtime and more computational power, and would be unsuitable for resource-constrained IoT devices like doorbells or thermostats.

D. *Important Feature Ranking Performance Evaluation*

Using the input perturbation ranking algorithm with XGBoost as explained in Section III, we obtained nine unique features that are ranked highly across all nine datasets. When these nine features were isolated for each dataset and the decision tree was run on a Raspberry Pi 4 with 8 GB RAM, we found that the detection accuracy was higher than 99.99%, whereas FPR was close to 0, as shown in Table V. It can be seen that the detection performance with only nine features is similar to that with 115 features in Table III. This indicates that these nine features are key in distinguishing between benign and malicious samples, irrespective of the type of IoT devices. Table V also shows the training and prediction times for datasets with only these nine features. Comparing these results in Table III with 115 features and Table V with only nine features, it can be seen that the training and prediction times are much smaller for decision trees with only nine features. For example, the average training and prediction times with nine features are 12.851 seconds and 0.021 seconds respectively, whereas they are 161.821 seconds and 0.18 seconds with 115 features. It can be seen that when the number of input features is reduced from 115 to nine, the speed of training for decision trees is more than 12 times faster, whereas the speed of prediction is more than eight times faster.

We also applied the decision tree with nine most important features to the combined dataset and ran the experiments in a MacBook Pro with 8GB RAM. The results are shown in Table IV for the row of "Decision Tree (9 Features)". We found that the training time is 46.93 seconds, and that the prediction time

Table V*: Detection and runtime performance of decision trees using nine most important features*

Dataset	Training Time (seconds)	Prediction Time (seconds)	TPR (%)	FPR (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Danmini Doorbell	13.707	0.023	99.999	0.03	99.998	99.998	99.999	99.999
Ecobee Thermostat	20.307	0.021	99.999	0.038	99.998	99.999	99.999	99.999
Ennio Doorbell	7.407	0.022	100	0.025	99.997	99.997	100	99.998
Philips B120N10 Baby Monitor	23.078	0.031	99.999	0.009	99.998	99.998	99.999	99.999
Provision PT737E Security Camera	8.823	0.019	99.999	0	99.999	100	99.999	99.999
Provision PT838 Security Camera	13.257	0.022	99.998	0.01	99.997	99.999	99.998	99.998
Samsung SNH1011N Webcam	4.135	0.011	100	0	100	100	100	100
SimpleHome XCS71002WHT Security Camera	12.402	0.02	99.999	0	99.999	100	99.999	100
SimpleHome XCS71003WHT Security Camera	12.543	0.022	100	0.026	99.999	99.999	100	100
AVERAGE	**12.851**	**0.021**	**99.999**	**0.015**	**99.998**	**99.999**	**99.999**	**99.999**

is 0.078 seconds. It can be seen that this is a great improvement in runtime performance when compared with the decision tree using all 115 features, while the detection performance remains almost the same. Therefore, with such reduced computational requirements, it becomes possible to run the botnet detection on resource-constrained IoT edge devices, by utilizing the decision tree with nine most important features.

VI. Conclusions and Future Work

Our method has been designed to act as an initial line of defense for IoT edge devices that usually have low memory and computation power. Our decision tree algorithm is a supervised machine learning method that is able to perform classification with high accuracy, low false positive rate, and small amount of time. In this paper, we have discussed the usage of decision trees and the ability to control the tradeoff between performance and computation time using maximum tree depths. We have also evaluated the detection and runtime performance of our method against a deep-learning based method (*i.e.*, Kitsune) and other widely used machine learning methods in detail, which showcases the efficiency of our algorithm. An important contribution of our research is to apply a feature ranking method (*i.e.*, input perturbation ranking algorithm with XGBoost) that produces nine most important features. These features, when segregated from network traffic, can help in identifying normal and malicious IoT traffic with high accuracy and less runtime on resource-constrained devices. Hence, it shows that our approach is efficient and cost-effective, especially as a first line of defense in the IoT environment.

As our on-going work, we plan to extend the study to adversarial example attacks [15], [16] against machine learning methods proposed in this work or other work for IoT. Given the most important features identified for IoT botnets, the questions arise: Is it possible for attackers to manipulate these important features in order to avoid the detection from machine learning methods? If possible, how can we defend against them?

Acknowlegment

This work was supported in part by 2020 Purdue University Fort Wayne Collaborative Research Grant.

References

[1] Wikipedia, Internet of Things [Online]. Available: https://en.wikipedia.org/wiki/Internet_of_things (October/2021 accessed).

[2] Radware, A Quick History of IoT Botnets [Online]. Available: https://blog.radware.com/uncategorized/2018/03/history-of-iot-botnets/ (October/2021 accessed).

[3] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection," in *Proc. of Network and Distributed Systems Security Symposium (NDSS) 2018.*

[4] R. Tang, Z. Yang, Z. Li, W. Meng, H. Wang, Q. Li, Y. Sun, D. Pei, T. Wei, Y. Xu, and Y. Liu, "ZeroWall: Detecting Zero-Day Web Attacks through Encoder-Decoder Recurrent Neural Networks," in *Proc. of IEEE INFOCOM 2020*, Toronto, ON, Canada, July 2020.

[5] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, D. Breitenbacher, A. Shabtai, and Y. Elovici, "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, Jul.-Sep. 2018, pp. 12-22.

[6] Q. Niyaz, W. Sun, A. Y. Javaid, and M. Alam, "A Deep Learning Approach for Network Intrusion Detection System," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, Feb. 2018, pp. 41-50.

[7] A. Whittaker, IoT gets a machine learning boost, from edge to cloud [Online]. Available: https://www.raspberrypi.org/blog/iot-gets-a-machine-learning-boost-from-edge-to-cloud/ (October/2021 accessed).

[8] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, M. Kallitsis, D. Menscher, Z. Durumeric, D. Kumar, C. Seaman, J. A. Halderman, L. Invernizzi, C. Lever, Z. Ma, J. Mason, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *Proc. of the 26th USENIX Conference on Security Symposium*, Aug. 2017, pp. 1093–1110.

[9] K. Angrishi, "Turning Internet of Things (IoT) Into Internet of Vulnerabilities (IoV): IoT Botnets," *arXiv:1702.03681*, Feb. 2017.

[10] N-BaIoT Data Set [Online]. Available: https://archive.ics.uci.edu/ml/datasets/detection_of_IoT_botnet_attacks_N_BaIoT# (October/2021 accessed).

[11] J. Heaton, "Feature Importance in Supervised Training," *Predictive Analytics and Futurism Newsletter*, No. 17, April 2018.

[12] J. Heaton, Application of Deep Neural Networks. [Online]. Available: https://github.com/jeffheaton/t81_558_deep_learning (October/2021 accessed).

[13] A. Geron, *Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow,* 2nd edition, O'Reilly Media, 2020.

[14] XGBoost Documentation [Online]. Available: https://xgboost.readthedocs.io/en/latest/ (October/2021 accessed).

[15] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial Examples: Attacks and Defenses for Deep Learning," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 30, no. 9, Sept 2019, pp. 2805-2824.

[16] H. Qiu, T. Dong, T. Zhang, J. Lu, G. Memmi, and M. Qiu, "Adversarial Attacks against Network Intrusion Detection in IoT Systems," *IEEE Internet of Things Journal,* vol. 8, no. 13, July 2021, pp. 10327-10335.