IOT Traffic Malware Detection Using Machine Learning Models

Karina Chichifoi

Master in Computer Engineering - University of Bologna

A.Y. 2022-2023

Abstract

In this research report, we present a study on the effectiveness of using machine learning models for detecting malware in Internet of Things (IoT) devices. Specifically, we trained and tested random forest and support vector machines (SVMs) models on the IoT23 dataset, which consists of 23 captures of IoT network traffic containing both benign and malicious traffic. The results of our experiments show that the proposed machine learning models are effective at detecting malware in IoT devices, with the random forest model achieving the highest performance in terms of accuracy and F1 score. Overall, our findings suggest that machine learning can be a promising approach for detecting malware in IoT devices, however some risks related to model and issues with data labelling should be considered.

1 Introduction

Internet of Things (IoT) devices have become an integral part of our daily lives, connecting a wide range of devices and objects to the internet, including appliances, vehicles [1], and healthcare [2] devices. As the number of IoT devices continues to grow, so does the risk of cyberattacks on these devices.

Cyberattacks on IoT devices can have serious consequences, including loss of sensitive data, financial damage, and even physical harm. For example, in 2016, a distributed denial-of-service (DDoS) attack was launched against DNS provider Dyn, which resulted in major websites such as Twitter, Netflix, and Reddit becoming unavailable in North America and Europe. The attack was conducted using a large number of Internet of Things (IoT) devices (e.g. printers, IP cameras, baby monitors), that had been infected with Mirai malware. Each device became part of a botnet that was used to launch the DDoS attack [3].

The first half of 2022 got off to a rough start: in January, IoT malware attacks had already doubled from the December immediately preceding it. Persistently high attack volumes over the next five months resulted in a total of 57 million IoT malware attacks in the first half of 2022, a 77% increase year to date [4].

The goal of this project is to purpose a new approach based on machine learning, which detects cyberattacks against IoT devices, to ensure the security and reliability of these devices. The source code is available at this repository: `https://github.com/TryKatChup/ML-IOT-malware-analysis`.

2 The IoT-23 dataset

For this task, we choose the IoT-23 Dataset to train new models, as it offers a well documented large dataset suitable for training machine learning algorithms. It consists of 23 different scenarios of different IoT network traffic; in particular, these scenarios are divided into 20 network captures from infected IoT devices and 3 benign network captures. The IoT devices used in this dataset are a Philips HUE smart LED lamp, Amazon Echo and Somfy smart doorlock. Each scenario is described by a PCAP file and an associated `conn.log.labeled` Zeek file [5].

Zeek is an open-source network security tool used to monitor and analyze network traffic for potential threats. It detects malware, intrusions, and other malicious activity, by analyzing network

packets and log files in real-time. [6] For this project we need only `conn.log.labeled` Zeek files, each one which consists in 23 columns.

The authors detonated specific malware in a Raspberry Pi, which became patient zero in the infection chain. The malware samples spread to the IoT devices, which are connected to the same network as the Raspberry Pi. The IoT devices share the same malwares as compromised botnet computers, of which the most common are Mirai, Torii and Okiru. There have also been cases where the devices were directly connected to "Command and Control" (C&C) servers, which is a different type of malicious connection flow.
Each feature is described below:

- `tf`: capture timestamp, expressed in Unix Time
- `uid`: capture id
- `id_orig.h`: source of attack IP address
- `id_orig.p`: source of attack port
- `id_resp.h`: IP of the IoT device
- `id_resp.p`: IoT device port
- `proto`: transport layer protocol of connection
- `service`: dynamically detected application protocol, if any
- `duration`: the amount of time data traded between the IoT device and the attacker
- `orig_bytes`: the amount of data sent to the IoT device
- `resp_bytes`: the amount of data sent by the IoT device
- `conn_state`: the state of the connection
- `local_orig`: whether the connection originated locally (`True`) or remotely(`False`); if `Site::local nets` value is empty, it is always unset
- `local_resp`: see `local_orig`
- `missed_bytes`: number of missed bytes in a message
- `history`: the history of the state of the connection
- `orig_pkts`: number of packets being sent to the IoT device
- `orig_ip_bytes`: number of bytes being sent to the IoT device
- `resp_pkts`: number of packets being sent from the IoT device
- `resp_ip_bytes`: number of bytes being sent from the IoT device
- `tunnel_parents`: the id of the connection, if tunnelled
- `label`: the type of capture, benign or malicious
- `detailed_label`: if the capture is malicious, the type of capture

3 Preprocessing

As we have 48 GB of Zeek log files we firstly use:

- `zat` python library to convert Zeek log files in a Pandas dataframe structure [7].
- `Dask` for its high-level parallel collections e.g. DataFrames, Bags, and Arrays, which operate in parallel on datasets that may not fit into memory. As we use a MacBook Air M1 with 8 GB of RAM it is quite useful to somehow manage 48 GB and process them in different moments, without loading all data in RAM. [8]

Then, we removed the `tunnel_parents` column, as it is almost empty, and we merged `label` and `detailed_label` in a unique `label` column, and encoded it, as shown in Table 1.

We checked the total null values in a column and deleted the columns with only null values. We observed the dataset was quite unbalanced (see Table 1): for this reason, we removed the class 3, 4, 6, and balanced the dataset, with only 9000 sample per `label` class (total of 63000). Finally, the dataset is small enough to be loaded in RAM using the `Pandas` library.

Class	Categories of Malware	Total samples
0	- (not a malware)	30860691
1	C&C, C&C-FileDownload	22048
2	C&C-HeartBeat, C&C-HeartBeat-Attack, C&C-HeartBeat-FileDownload	34518
3	C&C-Mirai	2
4	C&C-Torii	30
5	DDoS	19538713
6	FileDownload	18
7	Okiru, Okiru Attack	60990711
8	PartOfAHorizonalPortScan, PartOfAHorizonalPortScan-Attack, C&C-PartOfAHorizonalPortScan	213853817
9	Attack	9398

Table 1: Label assignment and number of samples for each class

After completing the previously mentioned steps, statistical correlation was used on the dataset to remove information that was not relevant to the label column. To acquire this data, a correlation matrix (see Figure 1) was created for the training set. The correlation matrices use colors, ranging from blue to red, to show the strength of the correlations. Blue signifies a negative correlation while red represents a positive correlation. Gray indicates a weak or no correlation.

The following columns were eliminated: `ts`, `local_orig` `local_resp`, `service`, `uid`, `duration`, `orig_bytes`, `missed_bytes`.

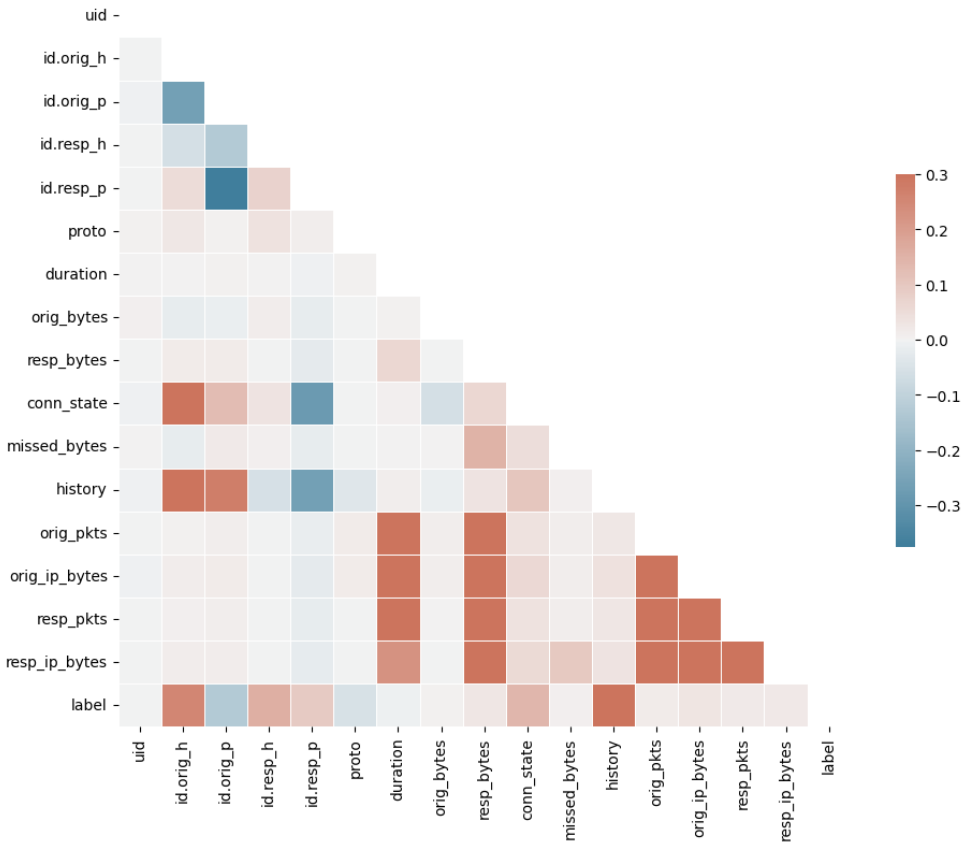

Figure 1: Correlation matrix on training set

4 Training machine learning models

For this task, we use the `scikit-learn` Python library, which provides a wide range of tools for tasks such as classification, regression, clustering, dimensionality reduction, model selection, and preprocessing [9]. It also includes a large collection of sample datasets and a range of evaluation metrics, which help testing and comparing different models.

4.1 Random Forest

Random forest is a supervised learning method used for classification and regression tasks. It is composed of a collection of decision trees, where each tree is trained on a randomly selected subset of the data. The final prediction made by the random forest is the average (for regression) or mode (for classification) of the predictions made by each individual tree. Random forest uses MDI (mean decrease in impurity, also called Gini) to calculate the importance of each feature for each node in the tree. The higher the decrease, the more important a given feature is.

One of the main advantages of random forest is that it can handle large datasets with high dimensionality, and it is relatively less prone to overfitting compared to a single decision tree [10].

4.2 Support Vector Machine

Support vector machines (SVMs) are a type of supervised learning algorithm that can be used for classification or regression tasks. Given a set of training examples, each labeled as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. An SVM model is a representation of the examples as points in space, mapped into a high-dimensional feature space so that examples from different categories are divided by a clear hyperplane. New examples are then mapped into that same feature space and predicted to belong to a category based on which side of the hyperplane they fall. [11]

In its simplest form, the SVM learns the separating hyperplane yielding the maximum margin between the hyperplane itself and the training examples: however, this behaviour could be problematic in the presence of outliers and could lead to overfitting. This can be prevented by choosing a suitable value for the hyperparameter C ($C > 0$). Such hyperparameter represents the "softness" of the margin: the lower the value of C, the softer the margin (i.e., more training examples will be misclassified, but the margin will be wider, leading in general to better accuracy on test examples).

Since the hyperplane is a linear function, the standard SVM is suited only to linearly separable datasets, which is not very useful in real-world scenarios. To enable the correct classification in the non-linear case, the data to classify is mapped by a certain function $\phi(\mathbf{x})$ into a new space, in which the data is linearly separable and thus in which SVM can be applied. However, computing the mapping $\phi(\mathbf{x})$ for every example $\mathbf{x}$ is computationally expensive; therefore, since only the inner product $\mathbf{x}_i \cdot \mathbf{x}_j$ is relevant (as far as fitting and classification are concerned), only the mapping of such product is considered (kernel trick):

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) \tag{1}$$

where is the kernel function. A common choice for K is the Radial Basis Function (RBF):

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2} \tag{2}$$

where $\|\mathbf{x}_i - \mathbf{x}_j\|$ is the Euclidean distance between the two examples $\mathbf{x}_i$ and $\mathbf{x}_j$, and γ is an hyperparameter defining how far the influence of a single training example reaches, with low values meaning "far" and high values meaning "close".

In case of multi-class classification, the `scikit-learn` library uses the One-versus-Rest method (OvR for short), which involves training a separate classifier for each class, with the samples of that class as the positive samples and all other samples as the negatives. During prediction, the classifier that has the highest confidence score for a given sample is chosen as the prediction for that sample.

4.3 Grid Search Cross-Validation

Grid search cross-validation is a method of hyperparameter tuning that involves training a model with a range of different hyperparameter values, evaluating the model's performance using cross-validation, and selecting the set of hyperparameters that performs the best.

In grid search cross-validation, a grid of possible hyperparameter values is specified, and the model is trained and evaluated using each combination of hyperparameters in the grid. For example, if the grid for a model includes the hyperparameters C and γ, and C has possible values $[0.1, 1, 10]$ and γ has possible values $[0.01, 0.1, 1]$, then grid search cross-validation will train and evaluate the model using all possible combinations of these values, resulting in a total of 9 different models. The combination of hyperparameters that yields the best performance on the cross-validation set is then selected as the best set of hyperparameters.

4.4 Results

In our experiments we firstly trained a linear SVM which, however, resulted in a sub-optimal accuracy of 72.80% and 72.58% on training and test sets, respectively: this may hint that the data is not linearly separable. In fact, the SVM with the RBF kernel (with parameters `C = 1000` and `gamma = 0.01`, chosen using grid search cross-validation) achieved a much higher accuracy on the test set (97.50%).

The best performance was obtained using a random forest model, resulting in an accuracy on the test set of 99.98% (with parameters `criterion = entropy`, `max_features = 0.5` and `n_estimators = 200`, chosen using grid search cross-validation).

Metrics (macro-average)	Random forest	Linear SVM	SVM with RBF kernel
Precision	99.98%	70.67%	97.50%
Recall	99.98%	72.50%	97.48%
F1-Score	99.98%	70.46%	97.47%

Table 2: Macro-average metrics for each model

Random forest usually perform better in a more general setting. The heterogeinity guaranteed by each tree trained on different parts of the dataset, and the use of different features for every single decision make random forest more suitable with more features.

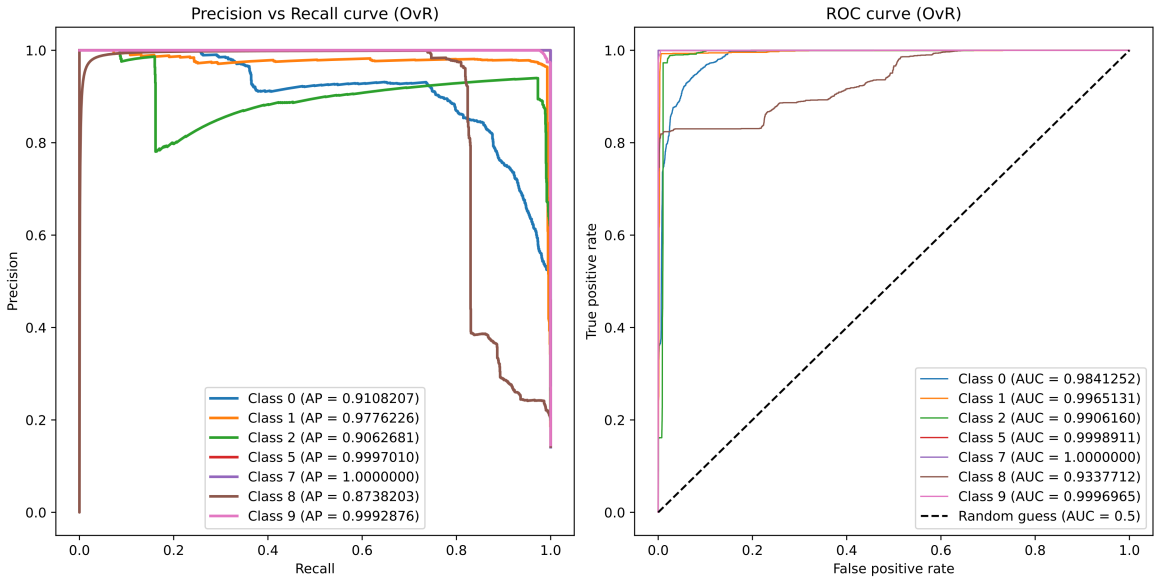

Figure 2: Receiver Operating Characteristic of SVC with RBF kernel

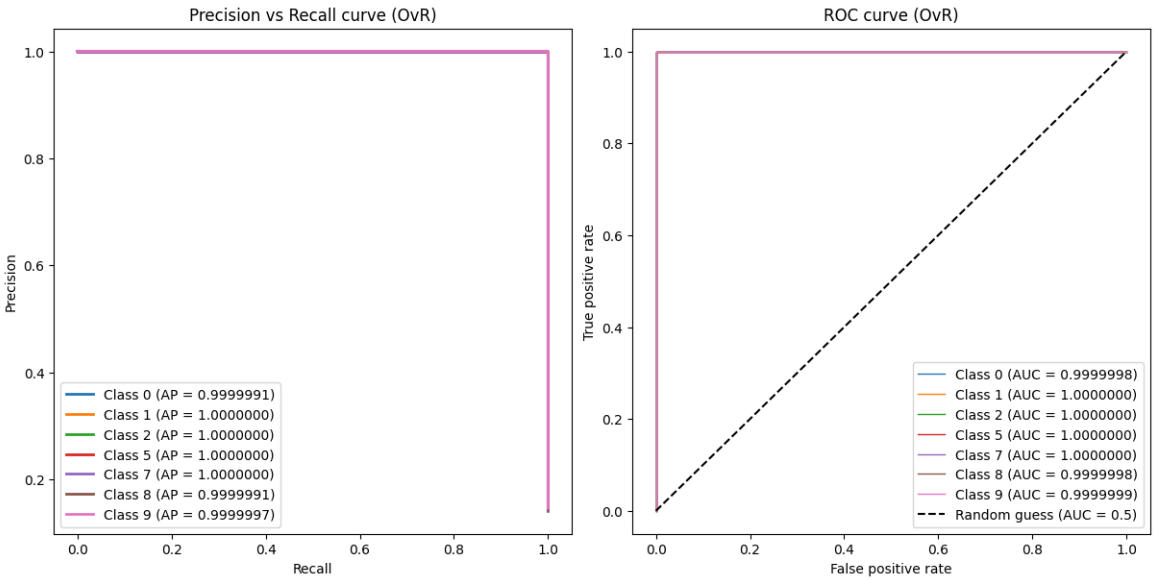

Figure 3: Receiver Operating Characteristic of random forest

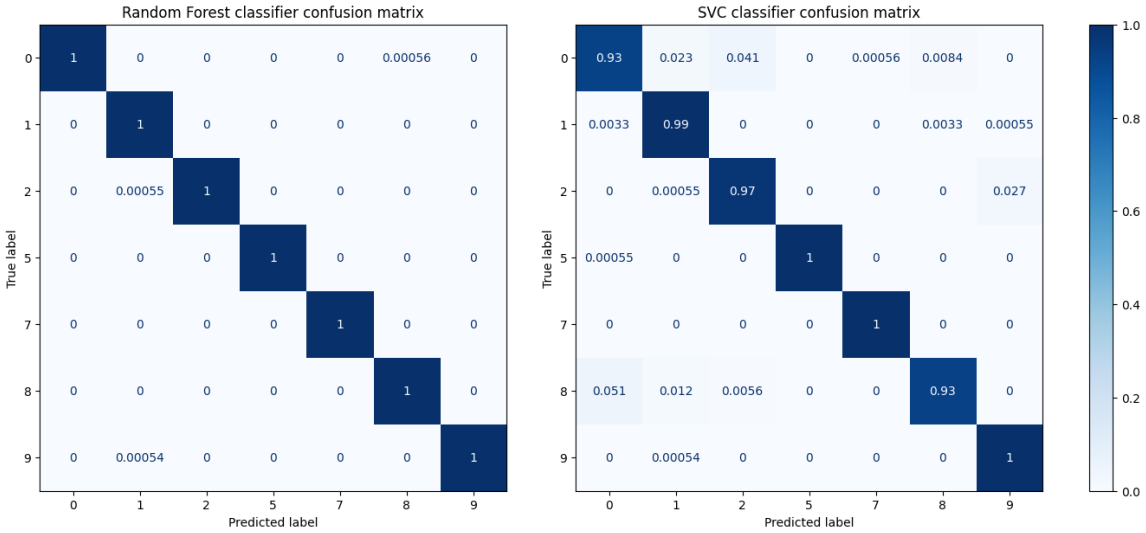

Figure 4: Random forest (left) vs SVC with RBF kernel (right) confusion matrix

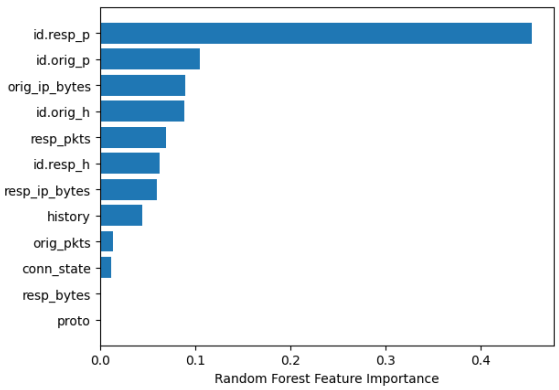

Figure 5: Most important features learnt by random forest model

5 Possible applications

There are few possible use case scenario:

- On a network device: the model could be executed on a network device (e.g., a router, firewall, or intrusion detection system) to analyze traffic in real-time as it flows through the network, doing both active and passive inspection.
- On a IoT device: as the model is really lightweight, some IoT devices could execute the model locally.

6 Conclusions

In conclusion, we discussed the use of machine learning for malware detection against IoT devices. We focused on a supervised learning approach, filtering our dataset and selecting the most important features. The results of our training indicate that the most promising model for this task is the random forest classifier, with an accuracy of 99.98%. Such model run either on a network device or an IoT device to prevent malicious flows. However, it should be noted that while machine learning approaches have the potential to significantly improve the detection of malware [12], they also present certain challenges. These include the need for large amounts of labeled data, which is expensive in matter of time and computational effort, and the ability of attackers to evade detection by evolving their tactics [13]. Possible future research should take into consideration these risks, and some cheaper alternative approaches, like unsupervised or semisupervised learning.

References

[1] X Krasniqi and E Hajrizi. "Use of IoT Technology to Drive the Automotive Industry from Connected to Full Autonomous Vehicles". In: *IFAC-PapersOnLine* 49.29 (Jan. 2016), pp. 269–274.

[2] Phillip A Laplante et al. "Building caring healthcare systems in the Internet of Things". en. In: *IEEE Syst. J.* 12.3 (2018), pp. 3030–3037.

[3] *Hackers Used New Weapons to Disrupt Major Websites Across U.S.* 2016. URL: https://web.archive.org/web/20221230203330/https://www.nytimes.com/2016/10/22/business/internet-problems-attack.html.

[4] *2022 Sonicwall cyber threat report.* 2022. URL: https://www.sonicwall.com/medialibrary/en/white-paper/mid-year-2022-cyber-threat-report.pdf.

[5] Sebastian Garcia, Agustin Parmisano, and Maria Jose. "IoT-23: A labeled dataset with malicious and benign IoT network traffic". Version 1.0.0. In: (2020). DOI: 0.5281/zenodo.4743746.

[6] *Zeek documentation.* URL: https://docs.zeek.org/en/master/.

[7] *ZAT - Zeek Analysis Tool.* URL: https://github.com/SuperCowPowers/zat.

[8] *The Dask Library.* URL: https://www.dask.org.

[9] *scikit-learn - Machine Learning in Python.* URL: https://scikit-learn.org/stable/.

[10] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32.

[11] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine Learning* 20.3 (Sept. 1995), pp. 273–297.

[12] Daniel Gibert, Carles Mateu, and Jordi Planes. "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges". In: *Journal of Network and Computer Applications* 153 (2020), p. 102526.

[13] Giovanni Apruzzese et al. "Addressing adversarial attacks against security systems based on machine learning". In: *2019 11th international conference on cyber conflict (CyCon)*. Vol. 900. IEEE. 2019, pp. 1–18.