Malware detection in IoT devices using Machine Learning

Bram van Dartel
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
b.vandartel@student.utwente.nl

ABSTRACT

The Internet of Things (IoT) is growing rapidly all over the world, while its security lacks behind. More than 30% of all the infections observed in mobile networks were targeted on IoT. Machine learning is suited for detecting malware on these, often unsupervised, devices and the results are promising. At this point, however, such detection in a single IoT node has not been done yet because IoT nodes often have weak processors. In this project, the possibilities of malware detection in a single IoT device are investigated by trying to scale machine learning algorithms such that a single IoT device can perform near real-time network traffic anomaly detection, marking packets as 'malware' or 'benign'. Using one of the machine learning algorithms, it is possible to implement the proposed program on an ESP32-chip that can classify data points from the IoT-23 dataset. When fully implemented, this could mean that, in the future, IoT devices will be able to check for themselves whether a network connection is part of a malware attack or if it is a 'normal' connection.

Keywords

Machine learning, IoT, IoT Malware, Malware detection IoT, IoT-23

1. INTRODUCTION

In 1999, the term "Internet of Things" was coined by Kevin Ashton, with which he meant that devices should gather data, instead of just people collecting data [1]. The current meaning of the Internet of Things, also known as IoT, is not that different. Nowadays, IoT is referred to as a network of small devices, that can sense the physical environment or act on the physical environment. An example of such an IoT device is a self-driving car, as it both senses the physical environment it is in and acts upon it.

The global market size of IoT grows rapidly [2] and with approximately 10 billion devices in 2021 [3], there are more of these devices than people in the world. Next to this increasing amount of the devices, the amount of IoT devices responsible for all infections observed in mobile networks went up from 16.17% in 2019 to 32.72% in 2020 [4].

Since usually IoT devices are not supervised [4], they are attractive as a target for malware. Often there is not even an option to monitor such a device. As the amount of observed infections in IoT devices grows, it becomes more clear that steps should be taken to combat these infections. One of the possible steps is to use machine learning to actively detect malicious incoming traffic. Since IoT generates a lot of data, this can be used as a training dataset for such a machine learning algorithm. One example of such a dataset is the IoT-23 dataset [5].

Using machine learning to detect malware in IoT is not something new. In earlier research on this IoT-23 dataset, machine learning was used to try to classify the type of malware [6]. Classifying the type of malware is not relevant for an IoT device, however, knowing whether a connection is malicious is. Letting each IoT device detect for itself whether a connection is malicious is something that has not been done yet. For this research, the following goal has been set:

Goal Discover whether it is possible to make machine learning scalable such that a single IoT device can detect malware attacks in real-time.

To achieve this goal, the following Research Questions have to be answered:

- **RQ 1:** Does a machine learning algorithm for detecting anomalies with only two labels (namely 'benign' and 'malicious') work better than algorithms where the malware is defined by multiple labels?
- **RQ 2:** What is the time performance and storage size of the malware classifier with only two labels?
- **RQ 3:** When comparing the newly proposed solution in **RQ 1** to current algorithms, which would better fit a single IoT node?
- **RQ 4:** Would it be possible to implement the solution found in **RQ 3** and, when looking at the time performance on the chip, would it still be a good solution?

By the end of the project, it is expected that this research contributes by checking whether using only two labels influences the accuracy and time performance of malware detection using machine learning. Furthermore, it contributes by actually showing a proof of concept that machine learning for malware detection can be run on IoT devices.

The structure of the rest of the research paper is as follows: An overview of related work of malware detection in IoT using machine learning will be given in Section 2. Then, in Section 3, the methodologies used to answer the aforementioned research questions will be discussed. Section 4 then will show the results that were obtained by following the methodology. By using the results, in Section 5 a conclusion will be drawn, after which Section 6 will contain the discussion and future work of this project.

2. RELATED WORK

In this section, an overview will be given of work related to IoT malware detection using machine learning.

In 2020, Ngo et al. [7] already researched the possibilities of malware detection in IoT and Android devices using machine learning. In this survey, they mainly focused on malicious files that could be executed. In the machine learning algorithms, static features were used of the files, such as the Opcodes or by running `strings` on a file. The accuracy of the different methods ranges between 85% and 99.8%. The RIPPER-classifier used for the ELF-header is interesting for comparison, since the classifying time of this algorithm is only 0.75 seconds, while the accuracy is 99.8%. It should be noted, however, that the time performance was not tested on a simple IoT device.

Diro and Chilamkurti [8] worked on a distributed attack detection scheme for IoT in 2017. For this research, they used deep learning to classify network traffic on an IoT device. They created so-called distributed fog nodes which are used for model training. Here, the deep model had a precision of 99.36% (on benign data) for 2-class detection and 99.52% (on benign data) for 4-class detection. Diro and Chilamkurti used the NSL-KDD [9] dataset for their model.

Hasan et al. [10] have worked on the dataset provided by Pahl et al. [11] to classify network data according to eight different labels, seven of which are malware types and the eighth label for 'normal'. The dataset that was used, has 13 features with which they managed to get an accuracy of 99.4% using the algorithms Decision Tree, Random Forest and Artificial Neural Network. Looking at other metrics, Random Forest performs comparatively better. It should be noted that the data used in the research was based on a virtual environment.

In 2020, Stoian [6] already worked with the IoT-23 dataset [5] to find out the best machine learning algorithm to classify the different connections from the dataset. The result for the project was a precision of 99.5% with the Random Forest algorithm. In the comparison with other studies, Stoian showed that the results using the IoT-23 datasets are in line with what could be expected, given the other works. In this research, ten different labels were used.

3. METHODOLOGIES

In this section, an elaboration is given on how the research was performed.

3.1 Initial dataset

In this research, the IoT-23 dataset [5] is used. This dataset contains 20 captures of malware executed in IoT devices and 3 benign captures. The data was captured in the Stratosphere Laboratory at the CTU University in the Czech Republic. The dataset offers the captures both in pcap format, which is the raw data capture, and in a labelled file with 23 different features. For this research, the labelled files were used, since they contained all the data that was needed. The 20 different labelled files together form the dataset that can be used as input for a classifier. In total, this means that the size of this dataset is approximately 53 GB. The amount of data points, or packets, is 325,307,990 of which there are 294,449,255 malicious datapoints.

The dataset contains, next to benign packets, malware packets that were categorized in 14 different sub-labels. Since this research focuses on malware detection and not the classification of the malware category, the sub-labels of the IoT-23 dataset are not deemed relevant. The original dataset has 23 features that can be used to train a classifier. The features, together with a short description, are included in **Table 1**.

3.2 Modifying the dataset

To load the plain-text files in Python [12], some modifications had to be done to the dataset. Next to these modifications, some other changes had to be incorporated as well to change the dataset into a dataset that was useful for this research.

3.2.1 Loading the dataset

The first goal was to load the dataset consisting of different files into Pandas data frames [13] since these data frames are easy to work with. To load the different files, some modifications had to be done in the different text files. There were still comments and a short explanation in every single one of the files, so they had to be removed. Furthermore, the different columns received their name, so they could easily be worked with.

One thing that was noticed, later on, is that the divider of the columns is not always the same, meaning that all occurrences of three spaces after each other had to be replaced by the divider, which is the `tab`-character.

To train a classifier, later on, the different datasets had to be merged. This was done by loading all the different datasets into Pandas data frames and then merging them into one. The new data frame was then stored as a CSV file.

3.2.2 Formatting & Pre-processing the dataset

At this point, the whole dataset can be loaded from a CSV file into one single data frame. There are however some columns that have different types. For example, a '-' - symbol is used in integer columns to represent the number zero. This was replaced by modifying the data frame using Pandas' built-in functionalities.

Now, the features are manipulated to serve the goal of this research. First of all, Stoian [6] already discovered using statistical correlation analysis that the features `ts`, `uid`, `id.orig_h`, `local_orig`, `local_resp`, `missed_bytes` and `tunnel_parents` can be removed. This was due to the weak relationship between the feature and the label. Another reason was that of some columns, namely the last three mentioned, there was not enough data such that a correlation could be measured.

Before removing the `ts`-label, a new feature was created that was based on the timestamp; the difference between the timestamps, or the inter-arrival time. To retrieve this interval, the whole dataset was sorted based on the timestamp. Now, the difference could be calculated and stored in a new column `time_diff`.

Last, before going to the machine learning step, the different strings were translated to integers using the scikit-learn LabelEncoder [14]. This means that a table is created with an encoding from a number to a string so that every unique string corresponds with a unique number. Now that this encoding is done, machine learning can be

Name	Name in dataset	Type	Description
Timestamp	`ts`	Float	Timestamp of the packet
Unique Identifier	`uid`	String	Unique identifier of the packet
Origin host	`id.orig_h`	String	IP-address of the origin of the packet
Origin port	`id.orig_p`	Integer	Port-number of the origin of the packet
Response host	`id.resp_h`	String	IP-address of the responding host of the packet
Response port	`id.resp_p`	Integer	Port-number of the responding host of the packet
Protocol	`proto`	Integer	Protocol over which the data was sent
Service	`service`	String	Service of the data in the packet (e.g. HTTP, DNS)
Duration	`duration`	Float	Duration of the connection with the other host
Origin # of bytes	`orig_bytes`	Integer	Amount of bytes sent by origin
Respondent # of bytes	`resp_bytes`	Integer	Amount of bytes sent by respondent
Connection state	`conn_state`	String	State of the connection
Local origin	`local_orig`	Boolean	Did the connection originate locally?
Local response	`local_resp`	Boolean	Did the response originate locally?
# of missed bytes	`missed_bytes`	Integer	Number of bytes missing in the packet
History	`history`	String	History of the connection state
Originating packets	`orig_pkts`	Integer	Amount of packets originated from host
Originating bytes	`orig_ip_bytes`	Integer	Amount of bytes originated from host
Responded packets	`resp_pkts`	Integer	Amount of packets responded by host
Responded bytes	`resp_ip_bytes`	Integer	Amount of bytes responded by host
Tunneled parent	`tunnel_parents`	String	Unique ID of parent if packet was tunneled
Label	`label`	String	Classification of the packet (benign/malware)
Detailed label	`detailed_label`	String	If the label is malware, the type of malware, otherwise empty

Table 1. The initial features of the IoT-23 labelled dataset.

performed on the dataset and the research questions can be answered.

3.3 On answering RQ 1

To check whether a machine learning algorithm for detecting anomalies with only two labels works better than algorithms where the malware is defined by multiple labels, the results of the two different methods should be compared. To do this, first of all, the labelled file was adapted, so that only the labels 'benign' and 'malware' would be used. Then, different machine learning algorithms were used to be able to compare the results of using only two labels with the results of having multiple labels, as Stoian [6] did.

The used algorithms are considered the more 'simple' machine learning algorithms, meaning that they might have a higher chance of being able to run on a single IoT node. This is both storage-wise and speed-wise. Also, these three algorithms have been used in referenced work, meaning it is possible to compare the results of this research with results from other researchers.

To answer which of the two works better, a look was taken at the accuracy, precision, recall, and F1-score. These metrics are calculated by using the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) metrics. A comparison will be made between this project and the work of Stoian, since that is the only research that is currently done using the IoT-23 dataset.

3.3.1 Decision Tree

A Decision Tree is a tree with nodes and leaves. The nodes can be interpreted as if-statements, while the leaves contain the categories and thus are the result of the classification. When starting with the classification, one starts at the start-node and follows the tree via the edges to a leaf, like a flow-chart.

A Decision Tree is the computationally lightest algorithm of the three. This means that a small chip should be able to classify a data point. Decision Trees are not very robust but can handle errors in the dataset [15]. Besides, Decision Trees tend to over-fit on the training data, which can be prevented by using a more complex classifier, such as a Random Forest classifier, which has Decision Trees as a basis.

3.3.2 Random Forest

The Random Forest classifier uses Decision Trees as a basis for its classification. By computing multiple different Decision Trees and then taking an average, the Random Forest often has higher accuracy than a Decision Tree [16]. The problem of over-fitting is overcome since the trees picked is randomised in the algorithm. Since a Decision Tree is computationally light, a Random Forest does not become much more advanced. The number of trees used in the Random Forest does however have an impact on the size of the classifier and, next to that, have an impact on the time it takes to classify a data point. This means there is a trade-off between accuracy and the complexity of the algorithm.

3.3.3 Naïve Bayes

The Naïve Bayes algorithm is based on Bayes' Theorem, which is

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Using this formula, one can calculate what the probability is of a statement `A` given a statement `B` holds. In order to calculate this probability, the probability of statement `B` given statement `A` times the probability of statement `A` should be divided by the probability of `B`.

Naïve Bayes classifiers are, just like the tree-based algorithms, very fast. Also, these types of classifiers do not need a lot of training data to already make excellent predictions [17]. Last, but certainly not least, Naïve Bayes classifiers are computationally light, which also means they do not take up much storage space when saved.

3.3.4 Metrics

The following metrics were used to determine the performance of the classifier.

Accuracy
The accuracy is specified as the formula

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

and calculates the proportion of correctly classified points over the total amount of classified points. Higher accuracy means a better classifier.

Precision
Precision is defined as the ability of the algorithm to not label a negative sample as positive. It is defined as

$$precision = \frac{TP}{TP + FP}$$

Concretely, this means that precision is the ability of the algorithm to not label a legitimate packet as malware.

Recall
The recall metric is specified as

$$recall = \frac{TP}{TP + FN}$$

and constitutes the ability of the classifier to find all the positive samples, so packets, in the test dataset.

F1-score
Since the classification that is done is only on two labels, the true F1 metric can be used. It is the weighted average of the precision and recall, defined by

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

3.4 On answering RQ 2

To answer this second research question, **RQ 1** must be answered. The two options, namely the dataset with only two labels and the dataset with a label per type of malware, should be compared on the time they take to execute. A comparison was made between the options found after answering **RQ 1**. This was done by running the test subset of the dataset on the classifier and measuring the time. When the complete dataset was classified, the total time was divided by the number of points that were classified, to get an average time that the classifier takes per point.

Next to the time, it is also important to see what the size of the classifier is. Although training does not need to be done on the small IoT device, the classifier itself, however, needs to be stored on the device. The 'pickle' module in Python [18] can store a Python object, in this case, the classifier, in a file so that it can be retrieved at a later point. In such a pickle file the classifier can be stored and uploaded to the IoT device. The size of this file is measured as the classifier size.

3.5 On answering RQ 3

The third research question is used to get to the goal of the research. Now that **RQ 1** and **RQ 2** are answered, a comparison is made to find out which (if any) algorithm would work better for a single IoT node to detect an anomaly in the network traffic. This comparison is made based on both the metrics mentioned at **RQ 1**, the size of the classifier and the time performance of the algorithm, which is measured at **RQ 2**. A choice should be made here between the different options to determine which algorithm would fit a single IoT node best.

3.6 On answering RQ 4

The fourth and final research question is the moment of truth; would it be possible to implement this on an ESP32-chip [19]? This chip is one of the most common chips used in the IoT since it is low-cost, low-power and has both Wi-Fi and Bluetooth (Low Energy). If it is possible to implement the proposed solution at **RQ 3**, it would be a proof of concept. For this research question, the LOLIN32 [20] board has been used, since this board has 4MB flash memory, which is sufficient for the classifier. Furthermore, the board has a micro USB port, so it could be easily connected to a computer to upload code.

To implement the proposed solution, it should be checked first whether the size of the proposed classifier is not larger than the storage space on the ESP32. Since this is the case, the implementation can be continued. First, it was tried to implement the algorithm using MicroPython [21], which did not seem a viable option. This was due to the several extra modules that had to be installed, which took more storage space on the board than was available. Another method was used, namely using the Python module 'micromlgen' [22], which is made for the export of classifiers from Python to C, such that small devices can also run machine learning algorithms.

By using this library, it was possible to generate C-code for the Decision Tree classifier. Of the other classifiers used, only the Naïve Bayes would have fitted. Although the file size of the Decision Tree classifier now was bigger, namely 170KB, it still fitted easily on the board. By using the 'EloquentTinyML' library [23] written for Arduino, it was possible to write C-code such that datapoints, stored as an array, could be classified.

4. RESULTS

This section will go more in-depth on the results that were obtained by following the described methodology. First of all, the experimental setup will be discussed, followed by the analysis of the results retrieved from the research.

4.1 Experimental Setup

For the experimental setup, a dockerized virtual machine was used to run the classification. The operating system used is Ubuntu 20.04.2 LTS 64-bit. The processor used was a 2x8-core Intel E5-2640 v3, CPU @ 3.40 GHz. The memory of the machine was 768 GB RAM. Implementation has been done using Python 3.8.5 64-bit and the module 'Pandas' for loading in the data in data frames and generating correlation matrices. Furthermore, the module 'scikit-learn' has been used for the machine learning and metrics.

Furthermore, as mentioned earlier, the LOLIN32 [20] board has been used, which has the ESP32 chip on board with 512kB of RAM and a 240MHz dual-core processor. The flash memory on this board is 4MB.

4.2 Result analysis

4.2.1 Research Question 1

For **RQ 1**, it was important to first implement the machine learning algorithm with only two labels. To answer the question of whether a machine learning algorithm for detecting anomalies with only two labels works better than algorithms where the malware is defined by multiple labels, the results of Stoian [6] are also included. First of all, in **Table 2** the metrics per algorithm are shown.

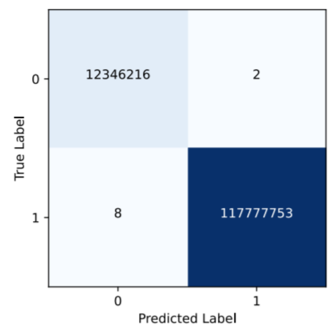

Figure 1. DT

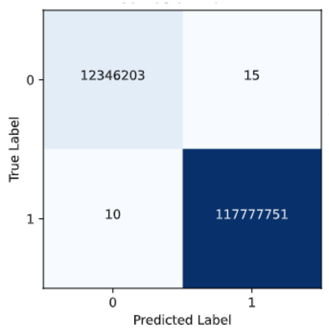

Figure 2. RF (n=10)

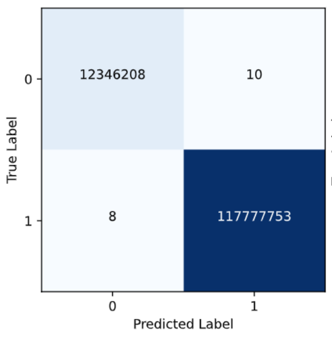

Figure 3. RF (n=100)

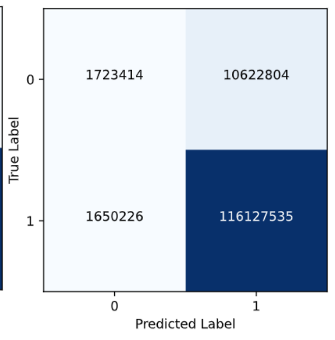

Figure 4. NB

As can be seen in this table, there is a negligible difference between the Decision Tree and Random Forest when looking at the metrics. All the metrics perform almost equally well, which is not surprising when there are only two labels included in the machine learning algorithm. Although it is negligible, it is interesting to see the confusion matrices in **Figures 1, 2, 3 and 4**. As can be seen, the Random Forest algorithm makes more mistakes in the classification than the Decision Tree does. This is interesting, since a Random Forest is an average of different Decision Trees, meaning that it should perform better than a single Decision Tree. The number of trees used, first 10 and then 100, did not make a significant difference in the confusion matrices. This points to the Decision Tree over-fitting on the dataset. Furthermore, the Naïve Bayes performs worse than the Decision Tree and Random Forest. Probably, this has to do with the fact that both the Decision Tree and Random Forest only do classification, while the Naïve Bayes algorithm gives back a probability of being in a class. The Naïve Bayes algorithm is not made to have classification as main-focus, which can be seen back in the metrics.

	DT	RF (n=10)	RF (n=100)	NB
Accuracy	0.999	0.999	0.999	0.906
Precision	0.999	0.999	0.999	0.916
Recall	0.999	0.999	0.999	0.986
F1-score	0.999	0.999	0.999	0.946

Table 2. Results of the metrics for the algorithms Decision Tree (DT), Random Forest (RF) and Naïve Bayes (NB).

When comparing the results to Stoian, see **Table 3**, it is clear that the metrics of this project are better than his, although it is just marginal. The conclusion that can be drawn here is that using only the labels 'benign' and 'malware' for the classification of the packets has better metrics than classification in 'benign' and the distinguished types of malware. It should be noted however, that this project could be seen more as an anomaly detector, instead of a real classifier. In the dataset, however, there is more training data on the 'malware' label than there is on the 'benign' label. The conclusion can be drawn in any case that the different metrics are for all but one algorithm better than when using multiple labels.

4.2.2 Research Question 2

The second question to answer is what the time performance and storage size of the classifier with only two labels is. First of all, some more explanation on the configuration of the Random Forest classifier is needed. The depth of a tree is automatically chosen by the scikit-learn module. The number of trees has been tried with two different settings; 10 trees and 100 trees. The accuracy when using 100 trees was negligible higher, namely smaller than a tenth per cent. On the other hand, the size of the classifier incremented from 3.2 MB to 31 MB, meaning that using more trees is not worth it when looking at the size, since there is only a limited amount of storage space on an IoT device.

Classifier	Size (KB)	Time (ns)
Decision Tree	63.710	173.44
Random Forest (n=10)	3313.0	990.36
Random Forest (n=100)	32090	9,447.0
Naïve Bayes	1.098	350.68

Table 4. Measurements of the size and time for the different classifiers. Random Forest has been included in both the 10 and the 100 trees variant.

In **Table 4**, the results of the average time that the classification of one point takes and the size of the classifier are included. As can be seen, the Decision Tree has the lowest size, while being one of the faster algorithms. It should be noted that the time of the classification is really fast, which is due to the experimental setup. When running this program on an IoT device, the time a classification takes will be significantly longer. This is due to the fact that the processor on the small board is not as fast as the processor used in the research environment. It is however good to already have an overview of the ranking in the speed of the classifiers.

4.2.3 Research Question 3

To check for malware in real time on a single IoT device, there are two limiting factors: the time performance and the size of the classifier. When, again, looking at **Table 4**, it becomes clear that the Random Forest with 100 trees certainly is not a good solution. The size of the classifier is too large for an IoT device and next to that, almost the same metrics performance can be guaranteed using the Random Forest of only 10 trees.

Then, when looking at **Table 2**, the Naïve Bayes algorithm does not perform that well in comparison with the other two remaining classifiers. Then, only the Decision Tree and Random Forest with 10 trees are left.

Looking at the metrics of the two algorithms, there is almost no difference, meaning that a look should be taken again at the size and time performance of the two classifiers. What is clear, is that the size of the Random Forest is more than 50 times as large as the Decision Tree and

Research	Best performing classifier	Accuracy	Precision	Recall	F1-score
Stoian [6]	Random Forest	1.00	0.995	1.00	1.00
This project	Decision Tree, Random Forest	1.00	0.999	1.00	1.00

Table 3. Comparison in metrics between this research and Stoian

that the time it takes for the Random Forest to classify a data point is more than 5 times as long. Since the influence on the performance metrics is not substantial, the Decision Tree classifier seems to be the best to implement on a single IoT device to check for malware in real time.

4.2.4 Research Question 4

As mentioned in the methodology, it was possible to implement the Decision Tree classifier on the ESP32 chip. The exported C-code of the classifier has been used, which has a size of 170KB. When running the classification process on 250 data points and then dividing the time it took by 250, it can be concluded that it takes approximately 0.24 ms per datapoint to be classified. It should be noted that the total time is only measured per hundredth of a second, meaning it is not very accurate.

Seeing only a short time and a small size that is taken up for the classifier on the ESP32-chip, the Decision Tree classifier still is a good choice to classify the data points given in real time. The program running on the IoT device that has the real IoT functionality for which the device was bought can, storage-wise, run easily next to the malware detection algorithm.

5. CONCLUSION

The goal of this research is to discover whether it is possible to make machine learning scalable such that a single IoT device can detect malware attacks in real-time. First of all, it is discovered that it is possible to do this. Using the Decision Tree classifier and an ESP32, it is possible to run the machine learning algorithm on a single IoT device. Next to only being able to run the algorithm on an ESP32, the classifier itself also has excellent performance metrics, meaning that it is also very accurate. It can be concluded that malware detection on a single IoT device is a promising possibility to reduce the number of infected IoT devices.

6. DISCUSSION

This discussion will be split into two different parts. First of all, the possible limitations of this project are discussed, after which some optimisations and future research will be mentioned.

6.1 Limitations

Some limitations can be found in this research. First of all, the encoding of the strings in the dataset should be mentioned. The origin host for example is an IP address that was interpreted as a string, meaning it was encoded according to it by the LabelEncoder. The LabelEncoder encoded them by giving every unique string a unique number. When a text occured multiple times, they were encoded as the same number. IP addresses give away a lot of information about the host, for example about the location but also about the Internet Service Provider (ISP). This means that if a lot of malware comes from a certain IP range or location, there is no way to detect that using this implementation. In future research, a solution may be found that takes the IP address into account in a better way than just encoding it as a string.

Next to the encoding of the strings, only three different algorithms in machine learning have been used. Although the time performance and metrics are excellent, it is always possible to improve. Now, the Decision Tree classifier has been chosen as the best algorithm to implement, but maybe another algorithm can be better. It should be noted, however, that not all the different types of algorithms are supported by the micromlgen module, meaning that probably a different library should be used.

6.2 Future research

In this research, some optimisations might be possible. The goal is to make the classifier as fast as possible, with a small size, while also maintaining the highest metrics possible. In the correlation matrix in **Figure 5**, it can be seen that there are a lot of features that do not correlate much with the label. There are only two features that stand out in this matrix, namely the `id.resp_p`, which is the respondents port, and the `duration`, which is the duration of the connection. A possible optimisation could be to only use these two features to reduce the classifier's size and increase the speed. The question remains whether the metrics are as good as they are right now. The newly-added feature of the inter-arrival time did only have a negligible correlation with the label, meaning it was not very useful for the classifier.

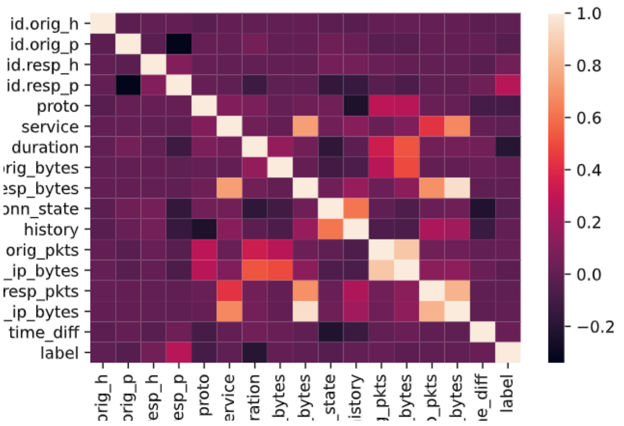

Figure 5. Correlation matrix of the complete dataset

Furthermore, now only the machine learning part has been implemented on an ESP32-chip. Future research could try to implement a whole program that gathers the internet packets received, extracts the features that are used and classify them. To really implement the real time malware detection on a chip this size, the whole process should only take a fraction of a second, which is still something that needs to be researched.

There are still a couple of data points that are misclassified. Although the count is negligible for the Decision Tree and Random Forest, it is interesting to see whether all these points are from the same type of malware. It might be the case that the algorithm finds it (just a bit) harder to classify malware from a certain type, but also to make such conclusions there should be extra research diving into this topic.

7. REFERENCES

[1] A. Kevin, “That ’ Internet of Things ’ Thing,” *RFiD Journal*, p. 4986, 2010.

[2] Lionel Sujay Vailshery, “Forecast end-user spending on IoT solutions worldwide from 2017 to 2025,” 2021. [Online]. Available: https://www.statista.com/statistics/976313/global-iot-market-size

[3] A. Holst, “Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030,” 2021. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[4] Nokia, “Threat Intelligence Report 2020,” Tech. Rep., 2020. [Online]. Available: https://onestore.nokia.com/asset/210088

[5] A. Parmisano, S. Garcia, and M. J. Erquiaga, “A labeled dataset with malicious and benign IoT network traffic.” 2020. [Online]. Available: https://www.stratosphereips.org/datasets-iot23

[6] N.-A. Stoian, “Machine Learning for Anomaly Detection in IoT networks: Malware analysis on the IoT-23 Data set,” Ph.D. dissertation, University of Twente, 2020. [Online]. Available: https://essay.utwente.nl/81979/

[7] Q. D. Ngo, H. T. Nguyen, V. H. Le, and D. H. Nguyen, “A survey of IoT malware and detection methods based on static features,” *ICT Express*, vol. 6, no. 4, pp. 280–286, dec 2020.

[8] A. A. Diro and N. Chilamkurti, “Distributed attack detection scheme using deep learning approach for Internet of Things,” *Future Generation Computer Systems*, vol. 82, pp. 761–768, may 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167739X17308488

[9] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” in *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications.* IEEE, jul 2009, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/5356528/

[10] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. Hashem, “Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches,” *Internet of Things*, vol. 7, p. 100059, sep 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2542660519300241

[11] M.-O. Pahl and F.-X. Aubet, “All Eyes on You: Distributed Multi-Dimensional IoT Microservice Anomaly Detection,” *IEEE*, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8584985

[12] “Python.” [Online]. Available: https://www.python.org/

[13] “pandas - Python Data Analysis Library.” [Online]. Available: https://pandas.pydata.org/

[14] “scikit-learn: machine learning in Python.” [Online]. Available: https://scikit-learn.org

[15] L. Rokach and O. Maimon, *Decision Trees.* Boston, MA: Springer US, 2005, pp. 165–192. [Online]. Available: https://doi.org/10.1007/0-387-25465-X_9

[16] L. Breiman, “Random Forests,” Tech. Rep., 2001.

[17] H. Zhang, “The Optimality of Naive Bayes,” Tech. Rep. [Online]. Available: www.aaai.org

[18] “pickle — Python object serialization.” [Online]. Available: https://docs.python.org/3/library/pickle.html

[19] “The Internet of Things with ESP32.” [Online]. Available: http://esp32.net/

[20] “LOLIN D32 Pro — WEMOS documentation.” [Online]. Available: https://www.wemos.cc/en/latest/d32/d32.htmlhttps://docs.wemos.cc/en/latest/d32/d32_pro.html

[21] “MicroPython - Python for microcontrollers.” [Online]. Available: https://micropython.org/

[22] “GitHub - eloquentarduino/micromlgen: Generate C code for microcontrollers from Python’s sklearn classifiers.” [Online]. Available: https://github.com/eloquentarduino/micromlgen

[23] “GitHub - eloquentarduino/EloquentTinyML: Eloquent interface to Tensorflow Lite for Microcontrollers.” [Online]. Available: https://github.com/eloquentarduino/EloquentTinyML