

HAMAD BIN KHALIFA UNIVERSITY
COLLEGE OF SCIENCE AND ENGINEERING

HARPY

Detection of Anomalous Deployment of IoT Devices

BY

Mohamed Amara

Omar Elshal

Selman Tabet

Senior Design Project Final Report

Submitted to the Faculty of
College of Science and Engineering

In Partial Fulfillment

of the Requirements

for the Degree of

Bachelor of Science in Computer Engineering

December 2020

© Mohamed Amara, Omar Elshal, Selman Tabet. All Rights Reserved

COMMITTEE

The members of the Committee approve the Project described in this document as defended on date [December 28, 2020]

[Dr. Ala Al Fuqaha]

Project Supervisor

[Dr. Jens Schneider]

Chair of the Committee

[Dr. Samir Brahim Belhaouari]

Committee Member

[Dr. Mohamed M. Abdallah]

Committee Member

Abstract

IoT devices have been growing in popularity in recent years, and with it comes issues regarding security and safety. Such devices can come take on many different forms and functions, some of which can behave in a malicious manner. To address this an application was made that gives network administrators a clear view of IoT devices in the network, powered by a Machine Learning algorithm to classify different devices. The classification algorithm was trained on data collected through previous research as well as the team's own internally collected data.

Contents

Abstract	ii
List of Figures	vii
Acknowledgements	ix
1 Specifications	10
2 Project Design & Structure	12
2.1 Work Package 1: Software	13
2.2 Work Package 2: Software	13
2.3 Work Package 3: Software	14
2.4 Work Package 4: Engagement	14
2.5 Work Package 5: Engagement	14
2.6 Work Package 6: Engagement	14
2.7 Work Package 7: Hardware	15
2.8 Work Package 8: Hardware	15
2.9 Work Package 9: Hardware	15
2.10 Work Package 10: Writing Final Report	15
3 Stakeholders	16
3.1 Input from Stakeholders	17
3.1.1 Siemens	17
3.1.2 Ministry of Interior in Qatar	18
4 Test Devices and Procurement	18

5	Literature Review	20
5.1	IoT Devices Recognition Through Network Traffic Analysis	20
5.2	Automatic Device Classification from Network Traffic Streams of Internet of Things	21
5.3	Investigating the Dark Cyberspace: Profiling, Threat-Based Analysis and Correlation	22
5.4	Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics	23
5.5	Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity	24
6	Methodology and Experimental Setup	26
6.1	Joy	26
6.2	The Sleuth Kit	28
7	Input System	30
7.1	Joy Shell Script	30
7.2	Feature Extraction Scripts	33
7.3	New Data Acquisition	36
7.4	Concerns and potential improvements	44
7.4.1	New Feature Overhead and Limitation	44
7.4.2	Router/Switch Requirements	46
7.4.3	Data-set-related Limitations	47
8	Machine Learning Software Program	49
8.1	Introduction	49

8.2	Machine Learning Program Functionality	49
8.3	Machine Learning Algorithm	51
8.4	Data Preparation	52
8.5	Machine Learning Model Creation and Training	59
8.6	Testing and Results Observations	61
8.7	Listening and Updating the Real-time Database	64
9	Output System	67
9.1	Overview	67
9.2	React Native	68
9.3	AWS	70
9.4	Firebase	71
9.5	Firebase vs AWS	72
9.6	The Single Board Computer	73
9.7	The Mobile Application	74
9.7.1	Limitations and Future Work	82
10	Conclusion	83
	References	84
	Appendix	86
A	Data Acquisition and Preprocessing	86
A.1	Shell Script	86
A.2	Harpy Shell Launcher	87
A.3	Feature Extraction Script	90

B Machine Learning Code	100
C React Native Code	110
C.1 Navigation	110
C.2 Firebase Configuration	111
C.3 Home Screen	111
C.4 List Screen	118
C.5 Device info page	123

List of Figures

1	Harpy Architecture	12
2	Joy Code Example	26
3	Joy Output After Execution	27
4	Sleuth Code Example	28
5	Sleuth Output After Execution	29
6	The original Shell Script.	31
7	The Launcher script.	34
8	The launcher info.	36
9	The first capture run	37
10	Error messages printed by Joy	38
11	Arbitrary packet inspection of a Linux cooked capture.	38
12	List of interfaces displayed on OpenWrt LuCI.	39
13	<code>br-lan</code> is available here, allowing us to capture all needed data. . . .	40
14	The second capture run.	40
15	Arbitrary packet inspection showing the correctly captured packets. .	40
16	Successful Joy output.	42
17	<code>tmux</code> for parallelizing workloads and keeping the capture process alive post logoff.	43
18	Sample of a device's extracted RDAP entry.	45
19	Random Forest algorithm functionality simplified	52
20	Snapshot of training data-set	53
21	Data-set file analysis	53
22	Open the CSV files	56

23	Drop any row of data with missing values	56
24	Drop the unnecessary features	57
25	Encode the categorical values	57
26	Create the Y data-set	58
27	Create the X data-set	58
28	Split the data-sets for training and testing	60
29	Import <code>RandomForestClassifier</code> from <code>sklearn</code>	60
30	Train the model	60
31	Test the model	61
32	Find the results	61
33	Performance analysis for the 5 minutes time scale model	62
34	Performance analysis for the 15 minutes time scale model	62
35	Performance analysis for the 30 minutes time scale model	63
36	Importance of each feature	64
37	Firestore Configuration	65
38	Listen and update - 1	66
39	Listen and update - 2	67
40	Listen and update - 3	67
41	Mobile app first page	75
42	Second page in the beginning	77
43	Second page after marking a safe device	79
44	Device info when it is unsafe	80
45	Device info when it is safe	81

ACKNOWLEDGMENTS

We would like to thank every individual that contributed to the success of this project. Especially, the Ministry of Interior in Qatar and Siemens Qatar for their valuable feedback. We would also like to thank Dr. Ala Al Fuqaha for guiding us throughout the journey.

1 Specifications

This project is going to focus on detecting malicious and rogue IoT devices in certain premises. With the advancements of the IoT devices, many of the new devices constitute huge threats against civilians. With characteristics like the micro size and the wide range of operations, the sheer number of IoT devices deployed worldwide and the vulnerabilities that were historically exploited time and time again, it became almost a necessity to have a device to protect the people from the dangers of having such devices planted in their private premises. A simple example of such a device is the IoT Micro Camera, which can be simply planted by an invader in someone's private area and record. Potential threats of an action like this is black mailing and collecting private information. Therefore, our main goal from this project is to provide security and assure privacy in households and private spaces like offices and stores.

Harpy's architecture (Figure 1) consists of a raspberry pi that is deployed in the premises of interest. It will collect the network traffic using traffic monitoring software such as Wireshark, then process the capture files with the Joy tool -which will be discussed later on the paper- in order to extract the important information. After that, data-sets will be sent to the server, and run the Long Short-term Memory + Convolutional Neural Network (LSTM-CNN) cascade model, to classify the IoT devices in the premises. After classification, the list of the IoT devices will be sent to the user on their mobile phone. That will be done through updating the database on Firebase and then updating the list on the mobile application.

The user will be able to see the devices installed and functioning around them. For future work; in the event of the discovery of a malicious device, we want to give the user the ability to take action, to jam the malicious device, for example.

To test our implementation we are going to set up a network of these IoT devices: (Samsung camera, TP Link Plug, Amazon Echo, and ChromeCast). Then we will use HBKU virtual machines to run the machine learning algorithms. The rest of the implementation will be the same as above.

Such a product can serve the community in many different sectors. Not only the private sector, but also offices and workspaces can use such a product to protect their privacy. Security nowadays play a big role in the success of any enterprise, and as potential threats become more dangerous, companies are spending more and more on privacy shielding.

We will try to design the product in a way that makes it as efficient and as user-friendly as possible. We will achieve that by providing the user with all the needed information about the intruding device, through the fastest and most accessible streams, like a mobile phone app for example.

Providing the user with accurate information is a very crucial part of this project, therefore, the machine learning aspect of it is going to be the foundation and building block. Higher accuracy can be achieved through intensive training and debugging, however, we aim to make the module self-taught allowing it to improve by time, relative to the amount of data it consumes.

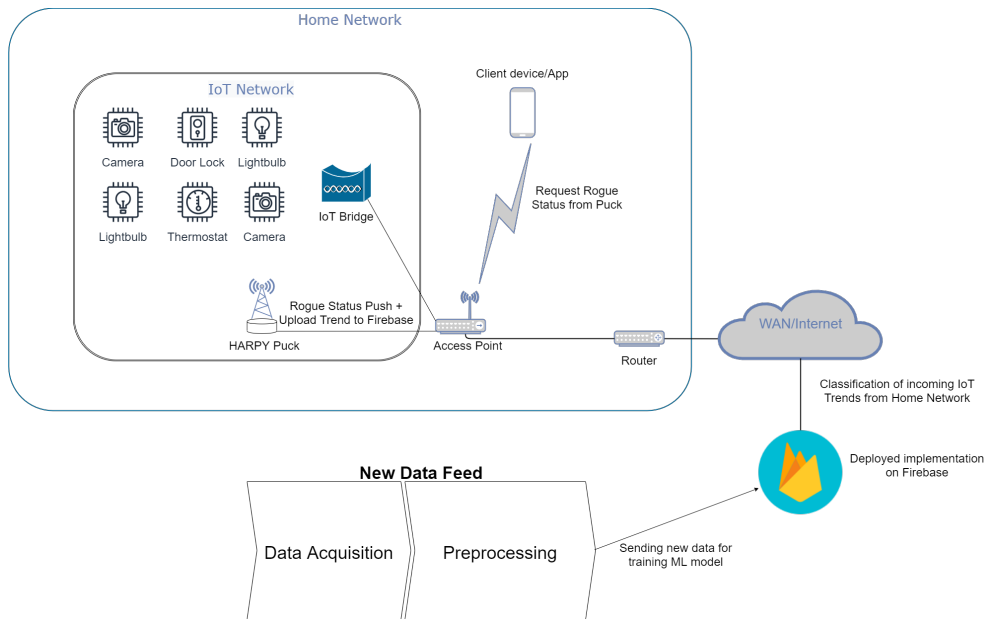


Figure 1: Harpy Architecture

2 Project Design & Structure

We decided to break our project into four phases. Each phase will consist of 3 work packages (WP), each assigned to one of the team members is going to be primarily responsible for it, along with another team member as the second worker for support. Except for the last phase where all team members will work on it together. The first phase will be focused on the software part of the project, mainly the development of the machine learning data classification algorithm, training the module and testing it. In the second phase we will focus on targeting our stakeholders and engaging with them. The purpose of this phase is to understand the needs of our stakeholders and the industry. We will try to conduct as many meetings with them as possible, and structure our hardware prototype based on the feedback we get from them. The third

phase will be dedicated to the hardware component of our project, which is basically the IoT network we are going to create to test our project. Finally, the last phase is going to be dedicated to writing the final report.

	10th April	10th, May	10th, June	10th, July	10th, August	10th, September	10th, October	10th, November	6th, December
WP1									
WP2									
WP3									
WP4									
WP5									
WP6									
WP7									
WP8									
WP9									
WP10									

Unstarted	
In progress	
Completed	

2.1 Work Package 1: Software

Goal: Collect data and organize it, develop initial tests.

Personnel: Selman and Omar will work jointly on this package.

Required Expertise: Selman will bring data science expertise and Omar will contribute experience in structuring the data.

Milestone: Prepare sufficient data for processing.

2.2 Work Package 2: Software

Goal: Programming and coding the machine learning module.

Personnel: Mohamed and Selman jointly

Required Expertise: Mohamed will write the code and develop the algorithm. Salman will contribute to the coding process and help with debugging.

Milestone: Develop an MVP of a program that identifies IoT devices.

2.3 Work Package 3: Software

Goal: Test the software and visualize results

Personnel: Omar and Mohamed jointly

Required Expertise: Expertise in graphing and visualizing data contributed by Omar.

Analyzing the results contributed by Mohamed.

Milestone: Complete training and testing

2.4 Work Package 4: Engagement

Goal: Research and scout for potential stakeholders.

Personnel: Selman.

Required Expertise: Solid understanding of the project and what it serves.

Milestone: Find at least three potential stakeholders in Qatar

2.5 Work Package 5: Engagement

Goal: Contact stakeholders and setup meetings with them.

Personnel: Omar

Required Expertise: Good communication skills.

Milestone: Take care of all engagement logistics

2.6 Work Package 6: Engagement

Goal: Prepare presentations and present to stakeholders in meetings.

Personnel: Mohamed

Required Expertise: Good presentation skills.

Milestone: Present to stakeholders.

2.7 Work Package 7: Hardware

Goal: Design and structure the IoT devices Network.

Personnel: Mohamed and Omar.

Required Expertise: Good understanding of the testing process.

Milestone: Set up the IoT network.

2.8 Work Package 8: Hardware

Goal: Connect the IoT network with the program developed in phase one.

Personnel: Mohamed and Selman

Required Expertise: Programming expertise and a solid understanding of the code developed.

Milestone: Establish the connection and start testing.

2.9 Work Package 9: Hardware

Goal: Test and debug.

Personnel: Selman and Omar

Required Expertise: Data visualization and debugging skills.

Milestone: Perform testing on the program, collect and analyze the results.

2.10 Work Package 10: Writing Final Report

Goal: Write the final report paper of the project

Personnel: All team members.

Required Expertise: Decent writing skills and a solid understanding of the project.

Milestone: Complete the final report.

3 Stakeholders

Our project is concerned about classifying rogue and potentially-malicious IoT devices within a vicinity. The first thing that comes to mind are security firms and enterprises that require security implementations to be sure that no malicious or unrecognized IoT devices are being placed in a restricted facility. Such firms include, but is are not limited to; major governmental bodies responsible for surveillance and covert operations (e.g. NSA, FBI, CIA, etc.), security firms as this can be a product for them to use in their own security services (e.g. G4S), even research centers like QCRI could be interested in using our project in a bigger scale solution that uses IoT device classification as a modular component.

As for smaller scale deployments, this would include regular households that would like to monitor its IoT network activity should there be an abnormal traffic behaviour caused by malicious devices or unrecognized devices that may not be known to exist by the household occupant.

An example of a scenario where a rogue IoT device might be of concern to all of the aforementioned stakeholders would be an IoT spycam that has been unknowingly installed as a part of a privacy-invading scheme, this would be an obvious thing to detect in an environment where IoT cameras are nonexistent as the stream would stand out apart from the rest. It would require a more sophisticated system to be able to identify such a rogue device on a network that already has several IoT cameras connected. This is especially common in large enterprises, which may value its security greatly, and therefore may be interested in investing in the HARPY system.

3.1 Input from Stakeholders

3.1.1 Siemens

We met with Mr. Tewfik Timeridjine, the head of digital grid in Siemens Qatar. We walked him through the project and then we got the following feedback from him.

Firstly, he suggested that adding the SBC would add more traffic to the network, and its activity would cause more congestion. So he thought that we should consider this when we look at our project from a commercial point of view. Another point he mentioned was that the time scales will create periodic traffic, which in turn will cause traffic congestion as well. To solve this issue, he suggested that we look into what is referred to in the industry as "detection-on-exception," which means that the device will only talk if there is a change in the information. Adding to that, he advised us to add a notification system to notify the user whenever a new device is detected, because he said that the machine should remind the user instead of the user reminding the machine. Another main point Mr. Timeridjine discussed with us was the universality of our product. It is extremely vital to ensure that our product works with all environments if we plan to introduce Harpy to the industry.

All in all, Mr. Timeridjine was happy about the outcome of our work and he was impressed by the simplicity of the interface, and the pipeline organisation of the project. He finally advised us that the next step would be thinking about how to protect after detection.

3.1.2 Ministry of Interior in Qatar

We contacted the Ministry of Interior in Qatar and we arranged a meeting with one of their representatives. We presented our project to him briefly and showed the features of it and its potential. We asked for his advice and recommendations to improve the project.

He mentioned two main points. The first one is that we must define the assets in the area where our device will be deployed and feed this to the database as a best practice. His second point was that capturing packets is very consuming when it comes to the resources; it uses more power with the working time, especially for big enterprises where there are hundreds of IoT devices, and it can cause the network to be heavier because of the continuous capturing.

4 Test Devices and Procurement

In order to test the efficacy of the HARPY system, we would need to simulate its behavior using a set of devices that would interact with the SBC. Different devices of various types would be used as a way of challenging the setup's ability to successfully classify one IoT device type from another. It is worth noting that a certain budget is required for the procurement of these devices for the sake of the experiment. The details are shown in the list below. Note that those devices are selected to best match with the data-set acquired from a previous research on IoT device classification. Certain items are different from that of the ones used for the data-set acquired. That is due to unavailability or high prices required to acquire the exact items.

- Echo Dot (3rd Gen) w/Clock - \$60
- Kasa Smart Plug by TP-Link (2-Pack) - \$25
- Wyze Smart Cam - \$20
- WiFi Chromecast - \$30
- Google Nest Hub - \$90

In addition to these items, a member of the team happened to have several Philips Hue lights and a switch available for use in the testing phase.

5 Literature Review

5.1 IoT Devices Recognition Through Network Traffic Analysis

The team studied a conference paper titled “IoT Devices Recognition Through Network Traffic Analysis” [1] where the authors explain their work on identifying and classifying IoT devices based on their network traffic. In their paper, they presented multiple methods to process the traffic data. Their results achieved an astounding 99.9% accuracy rate.

Their method was to extract bidirectional flows described by features like the size of the first N packets sent and received, along with the interval times. The Network traffic collected by a Raspberry Pi is firstly split based on MAC addresses, then the bidirectional flows along with their timestamps and protocols are extracted from the files. They only keep the TCP flows because all devices use HTTP or HTTPS protocols. All the flows then get merged into a single data-set and reordered based on their timestamps. Finally they use a classification algorithm to filter the data and identify the devices. They introduced six different algorithms that would help us in our project, as we are planning to process our data through a machine-learning algorithm to classify the IoT devices. After a thorough testing phase, they found that the best algorithm was Random Forest. However, we can not fully rely on this because in their work they only tested a network consisting of only 4 IoT devices, whereas we are targeting a bigger sized network in our project.

In this paper the authors discussed very crucial points regarding the process of classification which our project heavily relies on, such as; the metrics used to assess the machine learning algorithm (precision, recall, and F1 score), the averaging technique, and the relation between the number of packets N and the accuracy of the results. This information will be very beneficial for us in the process and will help us in so many ways to accomplish the target of our project.

5.2 Automatic Device Classification from Network Traffic Streams of Internet of Things

It is of the essence for the algorithm to be scalable into larger network deployments with the ability to classify more than just 4 devices, note that the number of active IoT devices is projected to reach around 22 billion [2]. The problem with scalability is the potential computational cost, we have been looking out for the most computationally efficient algorithms, that is, the most accurate ones for the compute time required. A paper titled “Automatic Device Classification from Network Traffic Streams of Internet of Things” [3] proposes a method involving a Long Short-term Memory + Convolutional Neural Network (LSTM-CNN) cascade model. The method proposed can supposedly maintain a high classification accuracy without the restrictive constraints required by an algorithm such as Random Forest e.g. does not require matching hardware and firmware to have the same classification. So new models and vendors that are unfamiliar to the trained set can still be able to identify the semantic type of the device based on the network stream, which often has its content encrypted in SSL, and that is fine since the headers are all that is required by the model.

5.3 Investigating the Dark Cyberspace: Profiling, Threat-Based Analysis and Correlation

This paper [4] is written by researchers from Concordia University, Canada. Discussed in this paper: profiling the darknet traffic, threat analysis, and threats correlation. Darknet, as defined in the paper, is “a set of unallocated network addresses and communication ports that belong either to the public cyberspace or to a specific organization. Such unallocated space could be maliciously utilized to launch cyber attacks.” First, the researchers start by profiling the darknet traffic: differentiating darknet packets, identifying the major protocols that are used in darknet traffic, profiling darknet application protocols, studying source and destination distributions of IP classes in the darknet traffic, identifying the resolved domain names in darknets, pinpointing the destination ports, and performing darknet geo-localization. Then, they executed threat-based severity analysis by implementing two open-source NIDS (Network Intrusion Detection Systems) which are Snort and Bro. The NIDS used were configured with rule sets from the Sourcefire Vulnerability Research Team and The Bro Network Security Monitor. After feeding the NIDS with the darknet data, an outcome of different threats with their type and priority is given. Lastly, they study threats correlation which is out of our project scope. We contacted one of the authors and we had a chance to get a sample of the darknet collected data. We will use the data to check if the IoT devices will go rogue and behave maliciously similar to said data. Secondly, we plan on using Snort and Bro to check the data generated by the IoT network that we will have later, and figure out if the data has any threats or not.

5.4 Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics

This paper [5] discusses how to use certain characteristics within the packets, such as port numbers, protocols and domain names to identify a device's semantic type based on a training set involving a variety of devices, including cameras, weather stations, light bulbs, hubs and non-IoT devices. The paper demonstrates a multi-stage classification architecture that identifies devices using their packet streams and abstracts features from them. It accomplishes this by isolating the traffic in a capture file down to a single device of choice using a tool called Joy. It then takes the port numbers, domain names and cipher suites of each packet in the filtered stream and feed them into a Naive Bayes Multinomial Classifier (First stage), this would give out a class identification along with a confidence value associated with it.

For the second stage, a series of other abstractions based on the entire stream, rather than per-packet data, are taken. They include packet flow rate, volume and duration. Along with sleep time, DNS interval and NTP interval. All of these, combined with the output of the previous stage, are fed into a Random Forest classifier to output the final classification and confidence value for the device of interest, originally isolated via Joy.

5.5 Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity

Recent reports show that attackers continue to exploit insecure IoT devices to launch volumetric attacks in the form of DoS, DDoS, brute force, and TCP SYN/UDP flooding. IETF has recently proposed the Manufacturer Usage Description (MUD) framework, in order to avoid this issue. As a result, vendors and producers of IoT devices should specify the intended behavior before putting them out in the market. The MUD specification has been approved as an RFC that motivates producers to embrace this standard when competing with other manufacturers. This specification allows an operator to lock down the network traffic of the IoT device using access control lists (ACLs) derived from its MUD profile. Yet there are some attacks that can be launched on IoT devices while still conforming to their MUD profiles. In this paper [6] they only consider volumetric attacks that are not prevented by the MUD profile, since its ACLs simply allow or deny traffic, and there is no provision to limit rates. To solve this problem they developed a system that learns expected patterns of MUD-compliant behavior for each IoT device by monitoring its activity via a combination of coarse-grained (per-device) and fine-grained (per-flow) SDN telemetry at various time scales, and has the capability to detect volumetric attacks and the specific traffic streams that contribute to it.

To develop their system they used an OpenFlow switch, Faucet SDN controller, and MUD policy engine. Their solution was released to the public as open source to the community. The paper [6] also highlighted other approaches to solve this issue, such as the signature-based approach, which in turn was proven inefficient for addressing the and growing security issues that come with the proliferation of IoT devices. This paper suggests that anomaly detection holds promise as a way of detecting new and unknown threats. However, despite extensive academic research it has had very limited success in operational environments.

A valid MUD profile contains a root object called “access-lists” container that comprises several access control entries (ACE), serialized in JSON format. Access-lists are explicit in describing the direction of communication, i.e. from-device and to-device. Each ACE would match on source/destination port numbers for TCP/UDP, and type and code for ICMP [6]. The MUD specification also distinguishes local network traffic from internet communication. The authors of this paper only released the MUD profiles for 28 consumer IoT devices as of the date of the paper publication.

This paper [6] and the work demonstrated in it show big potential. It can be of a great help to the team as the MUD profiling system would make the classification process easier and faster, by cutting edges, and providing data that the team need in a simpler and more compressed package. This paper [6] discusses many other aspects but they were not mention here as they are completely out of the scope of the project.

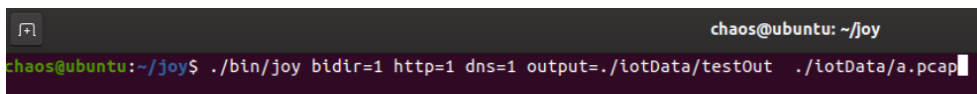
6 Methodology and Experimental Setup

6.1 Joy

The joy tool [7] essentially takes in multiple PCAP files, processes them and outputs a JSON-formatted summary of the network packets according to the parameters passed upon invocation. It can also capture packets using its “online mode” or when the interface is specified as an input parameter.

Joy provides many options on the sort of information that can be extracted from each packet, such as HTTP metadata, DNS queries, TLS, DHCP, SSH, IKE, etc. All through different data-related input parameters. This tool would enable us to take an IoT data-set and extract the necessary information required for the framework. The JSON structure allows easy data extraction which helps the developer to work on translating the data into a format that is compatible with the framework.

An example of a joy command is shown here:

A terminal window screenshot with a dark background. The prompt is 'chaos@ubuntu: ~/joy'. The command entered is './bin/joy bidir=1 http=1 dns=1 output=./iotData/testOut ./iotData/a.pcap'. The cursor is at the end of the command.

```
chaos@ubuntu: ~/joy
chaos@ubuntu:~/joy$ ./bin/joy bidir=1 http=1 dns=1 output=./iotData/testOut ./iotData/a.pcap
```

Figure 2: Joy Code Example

This takes a test file `a.pcap`, goes through all the packets, stitches together packets belonging to the same client-server connection session (`bidir=1`), and then reports HTTP (`https=1`) and DNS (`dns=1`) data, if any. Then finally, it dumps the JSON output in a specified `testOut` file using the output parameter, instead of using `stdout` to print the result. This would give us the ability to transfer the output of the joy tool for use in other components of the framework.

6.2 The Sleuth Kit

The Sleuth tool is a program that comes alongside Joy. First, we feed it with JSON or PCAP files, then choose our criteria and filters, then we get the desired output. The operations used are pretty much similar to SQL syntax.

One of the filters can select particular elements and exclude the others, which is the case in the `--select` command. An example of that would be: `sleuth --select packets[b]`. This selects only the bytes from the packets. Another one is the `--where` command which filters every object against this condition. The command `--dist` is used to calculate the distribution of the objects, number of appearances in the stream, and the total number of objects. Others are `--groupby`, `--sum`, `--no_stitch`, and `--fingerprint`.

All in all, the tool is fairly easy, pretty straightforward and is capable of analyzing network capture files, and gives a readable output that we can use in building our model.

This is a screenshot of the input:

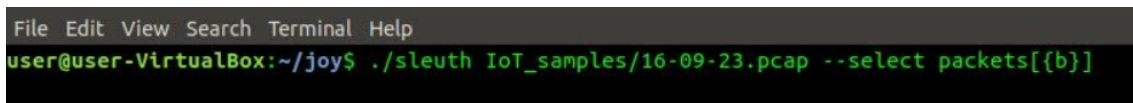
A screenshot of a terminal window with a dark background. The menu bar at the top shows 'File Edit View Search Terminal Help'. The prompt is 'user@user-VirtualBox:~/joy\$'. The command entered is './sleuth IoT_samples/16-09-23.pcap --select packets[{b}]'.

Figure 4: Sleuth Code Example

This is the command where we select only the bytes from the packets.

A snippet of the output:

```
File Edit View Search Terminal Help
{"packets": [{"b": 152}, {"b": 238}]}
{"packets": [{"b": 187}, {"b": 132}, {"b": 193}]}
{"packets": [{"b": 187}, {"b": 140}, {"b": 193}]}
{"packets": [{"b": 187}, {"b": 132}, {"b": 193}]}
{"packets": [{"b": 187}, {"b": 225}, {"b": 193}]}
{"packets": [{"b": 187}, {"b": 205}, {"b": 193}]}
{"packets": [{"b": 36}, {"b": 226}]}
{"packets": [{"b": 45}, {"b": 204}]}
{"packets": [{"b": 37}, {"b": 53}]}
{"packets": [{"b": 182}, {"b": 326}]}
{"packets": [{"b": 37}, {"b": 53}]}
{"packets": [{"b": 272}, {"b": 1448}, {"b": 181}]}
{"packets": [{"b": 37}, {"b": 53}]}
{"packets": [{"b": 221}, {"b": 265}]}
{"packets": [{"b": 30}, {"b": 488}]}
{"packets": [{"b": 48}, {"b": 48}]}

```

Figure 5: Sleuth Output After Execution

7 Input System

The initial Machine Learning model was trained using a data-set generated by an earlier research by a group of researchers from The University of New South Wales (UNSW), the data-set consists of network captures spanning a period of approximately one month, separated in chunks of one day long PCAP Capture files, therefore, for the sake of easier coding, the PCAP files were named `1.pcap`, `2.pcap`, `3.pcap`, etc. Each day is stored in one PCAP file. For other data-sets, this does not necessarily need to be separated into one-day intervals as they can instead be separated into smaller intervals e.g. three PCAP files for one day (8 hours each), or be combined to make bigger intervals e.g. two days in one PCAP file.

7.1 Joy Shell Script

The shell script in Figure 6 extracts network flow information via a tool named Joy.

```

PATH=$PATH:/sbin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/home/chaos/joy/bin

ethr_addr_list="d0:52:a8:00:67:5e
                44:65:0d:56:cc:d3
                70:ee:50:18:34:43
                f4:f2:6d:93:51:f1
                00:16:6c:ab:6b:88
                30:8c:fb:2f:e4:b2
                00:62:6e:51:27:2e
                e8:ab:fa:19:de:4f
                00:24:e4:11:18:a8
                ec:1a:59:79:f4:89
                50:c7:bf:00:56:39
                74:c6:3b:29:d7:1d
                ec:1a:59:83:28:11
                18:b4:30:25:be:e4
                70:ee:50:03:b8:ac
                00:24:e4:1b:6f:96
                74:6a:89:00:2e:25
                00:24:e4:20:28:c6
                d0:73:d5:01:83:08
                18:b7:9e:02:20:44
                e0:76:d0:33:bb:85
                70:5a:0f:e4:9b:c0
                08:21:ef:3b:fc:e3
                30:8c:fb:b6:ea:45
                40:f3:08:ff:1e:da
                74:2f:68:81:69:42
                ac:bc:32:d4:6f:2f
                b4:ce:f6:a7:a3:c2
                d0:a6:37:df:a1:e1
                f4:5c:89:93:cc:85
                14:cc:20:51:33:ea"

fileCo=1
while [ $fileCo -le 20 ]
do
    fileName="${fileCo}.pcap"
    echo Processing $fileName
    deviceCo=1
    for addr in $ethr_addr_list
    do
        joy output="json_files/${fileCo}_${deviceCo}.json"\
            bpf="ether host ${addr}"\
            bidir=1\
            dns=1\
            http=1\
            "pcap_files/${fileCo}.pcap"
        ((deviceCo++))
    done
    ((fileCo++))
done
echo Done

```

Figure 6: The original Shell Script.

The script iterates through each PCAP file, which consists of data captures each spanning one day, while iterating through a list of MAC addresses, outputting JSON flows for each device in each day. The JSON files end up being named in a format like this: `inputFileName_deviceNumber.json`

The `joy` command takes the following arguments:

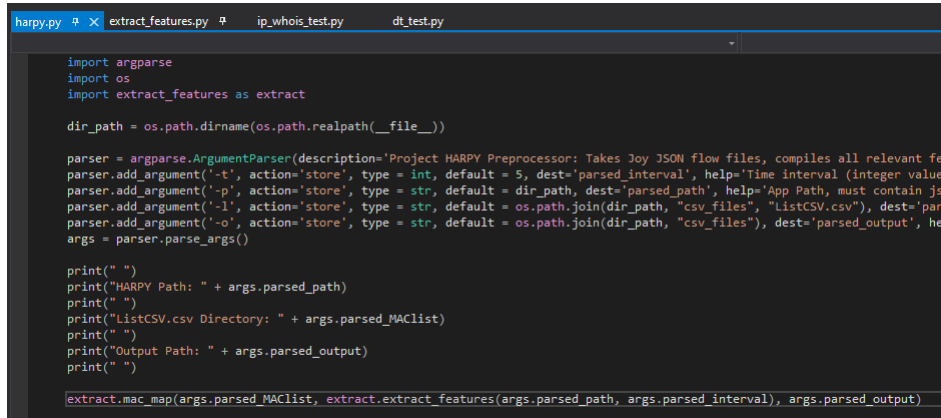
- `output` = the JSON file output
- `bpf` = Berkeley Packet Filter, can be used to filter the captures down to a single MAC address i.e. `ether host <mac_address>`
- `bidir` = Accounts for packets flowing in AND out (bidirectional).
- `dns` = Include DNS traffic info in the flows, which would be used to extract more features.
- `http` = Include HTTP traffic info, it would also be used to extract more features.
- The last argument is the capture file, which would be the input for the Joy tool.

The shell dumps the output set of JSON files into a subfolder named `json_files`. Each JSON file consists of a set of flows, and each flow consists of flow-related information such as destination address and flow size, all of which would eventually be aggregated into features that are handed over to the machine learning model for classification. The JSON files would then be used for a Python script that extracts a set of features that would be used as the input for the machine learning model.

The original shell script and python module was provided by Dr. Ihab Mohammed from the Western Michigan University. The python module consists of code that extracts features from the output of the joy commands executed via the shell script previously discussed. Information including, but not limited to, the number of unique servers, transmission/sleep times and sizes of flows, are aggregated over pre-defined intervals for each device separately. The output would be a CSV file where each entry is the aggregation of the aforementioned features over a single interval for a given device. As of the date of writing this report, the Harpy shell launcher as well as feature extraction scripts, are available on GitHub on <https://github.com/selmantabet/harpy>.

7.2 Feature Extraction Scripts

The original script required manual code overwrites for different data-sets as well as different MAC lists since the relevant values were initially hard-coded into the Python extraction module. This Python code was heavily modified in the name of modularity, that is, the script needs to be something akin to a black box that can perform focused tasks and only those focused tasks, rather than doing everything from one script. Automation is a long-term goal that is kept in mind as Harpy was developed. The code was broken down to different function definitions that can be called via a different Python module that acts as the main Harpy shell launcher, shown in Figure 7.



```
harpy.py # X extract_features.py # ip_whois_test.py dt_test.py

import argparse
import os
import extract_features as extract

dir_path = os.path.dirname(os.path.realpath(__file__))

parser = argparse.ArgumentParser(description='Project HARPY Preprocessor: Takes Joy JSON flow files, compiles all relevant fea
parser.add_argument('-t', action='store', type = int, default = 5, dest='parsed_interval', help='Time interval (integer value,
parser.add_argument('-p', action='store', type = str, default = dir_path, dest='parsed_path', help='App Path, must contain jso
parser.add_argument('-l', action='store', type = str, default = os.path.join(dir_path, "csv_files", "ListCSV.csv"), dest='pars
parser.add_argument('-o', action='store', type = str, default = os.path.join(dir_path, "csv_files"), dest='parsed_output', hel
args = parser.parse_args()

print(" ")
print("HARPY Path: " + args.parsed_path)
print(" ")
print("ListCSV.csv Directory: " + args.parsed_MACList)
print(" ")
print("Output Path: " + args.parsed_output)
print(" ")

extract.mac_map(args.parsed_MACList, extract.extract_features(args.parsed_path, args.parsed_interval), args.parsed_output)
```

Figure 7: The Launcher script.

This new module takes arguments via the shell, where certain paths are to be defined. This would accommodate to a variety of environments with different file structures. It is worth noting that this module was made with back-end developers in mind, the end-users (see Stakeholders) are not supposed to directly interact with this and instead, preprocessing and data acquisition should require the least amount of manual human intervention and is best left automated by relying on system calls rather than manually-issued shell commands. The original structure required the original developer to manually redefine values for a specific data-set, and the changes discussed here are made as steps towards the full automation of Harpy’s preprocessing element. This was all done under the “Compartmentalization Update – V3.2”.

Major improvements to the feature extraction code were made; the set of extracted features was enhanced by the addition of a new attribute; the Registration Data Access Protocol (RDAP) record details from an IP address in a Joy flow. It is worth clarifying that RDAP is often referred to in the code by the legacy standard WHOIS and is interchangeably called WHOIS in this report as well. While inadvisable from

a viciously technical standpoint, it is done intentionally because the library used was named after the legacy standard, while the queries themselves were in fact, RDAP. They are just two different protocols achieving the same goal after all: querying resource registration data from different sources such as Domain Name Registries (DNRs) and Regional Internet Registries (RIRs).

The python library `ipwhois` was used to achieve this. A query is placed when inspecting each IP address and the responses are received in JSON format, after thorough inspection of different query responses, it was concluded that the key named `asn_description` often had the most descriptive string for a given IP, that is, the strings were deemed to be identifiable information linked to registered entities. So upon the receipt of the RDAP responses, each flow is marked with the `asn_description` string, and the number of flows containing the same IP addresses within a single interval is aggregated, the `asn_description` and the number of instances of flows are then inserted into the CSV fields as a key-value pair.

The initial runs with this enhanced feature extraction script took several hours to finish, this turned out to be caused by the overhead required to resolve bulk RDAP requests. Because of the more static nature of WHOIS/RDAP records, an approach towards optimization was to locally dump all resolved IP addresses with their corresponding `asn_description` values into a JSON file for use in future script runs. This part was completed under the “Compartmentalization and Optimization Update – V3.4”.

```
#
#
# PROJECT HARPY - HBKU Final Year Project
#
# Data Collection, Preprocessing and Launcher Developer: Selman Tabet
# Machine Learning Developer: Mohamed Amara
# Frontend and App Developer: Omar Elshal
# Supervisor: Dr. Ala Al-Fuqaha
#
#
# ---- Prerequisite file structure ----
# - Harpy directory // Main app directory (MANDATORY, most essential argument to be provided via CLI unless already cd'd to it)
# |_ /pcap_files/ // Capture files (MANDATORY - for Joy Tool Shell Script)
# |_ /csv_files/ // (Default) Output of the feature extraction script (MANDATORY)
# |   |_ ListCSV.csv // List of MAC addresses (Optional iff the absolute path was provided via CLI)
# |_ /json_files/ // Where Joy JSON output is stored (MANDATORY)
# |   |_ whois_record.json // For RDAP requests in extract_features.py (MANDATORY)
#
# DO NOT RUN UNLESS THE MANDATORY FILES AND FOLDERS IN THE FILE TREE ARE COMPLETE
#
#
#
# Changelog: HARPY V3.4.2 - Style Update.
#
# [STYLE]
# - Moved changelog to harpy.py
# - Additional harpy.py Launcher styling.
#
#
# [ROADMAP]
#
# * Next Beta branch, in anticipation of new datasets:
# ~ Have harpy.py output the number of MACs in ListCSV.csv, this would be handed over to extract_features as an argument that goes directly
# to the device_co for loop to avoid hardcoding values.
# ~ For the same reason previously stated, the number of capture files needs to be calculated so that the value in the days_co for loop may
# be dynamically adjusted for the files in /pcap_files/ directory.
# ~ New Python script may be developed to call cisco/joy for all MACs in ListCSV.csv, but this would limit the execution environment to POSIX
# due to the potential usage of os.system(joy <args>), unless the user has "joy" cmd(let) somehow defined on their NT system.
# ~ In the event that the WHOIS Record was found to be outdated, it shall be cleared and rebuilt to address concerns with outdated registries.
#
```

Figure 8: The launcher info.

7.3 New Data Acquisition

A set of IoT devices were procured specifically for the project, a Linksys router was borrowed from the university and was flashed with OpenWrt firmware. The router was used to connect all the devices together to form the LAN. And with the new firmware, the team could install packages and have full control over the router's configurations.

At first, port mirroring seemed like the most logical method of capturing new files; send all packets going in and out of the router into a designated port where a PC with Wireshark may be plugged in and collect all traffic, which sounds easily feasible on the surface. Due to the more complex nature of the router, it was not

as straightforward as it first seemed from internet research, this led to a prolonged halt to the progress where the team did not manage to get useful packets sniffed from all of the router's NICs, but instead it seemed like it only mirrored packets traveling through the Ethernet LAN ports instead of the desired Ethernet + Dual-band Wi-Fi collective. A team member reached out to several individuals through a variety of networks, including Reddit and AlRayyan TV's IT staff, seeking out for help on the matter.

Due to the time constraint and lack of responses at the time, the team adapted a different, less-than-ideal, but still effective method, which is installing `tcpdump` on the router and having it run on the router's side, while storing the capture files on a USB plugged into its USB3.0 port. The capture dump is then transferred to a computer on the LAN via SCP for further processing.

Configuring the router for this new method required setting up a partition for the 128GB USB drive, this guide describes the process step by step: <https://openwrt.org/docs/guide-user/storage/usb-drives>

After that, a simple `opkg install tcpdump` command was issued, and the team was off to the races. The first run was set such that a new file was created every 8 hours (argument `-G 28800` seconds), shown in Figure 9.

```
root@OpenWrt:~# tcpdump -i any -G 28800 -w /mnt/sdal/capture_%m-%d_%H-%M.pcap
tcpdump: listening on any, link-type LINUX_SLL (Linux cooked v1), capture size 262144 bytes
```

Figure 9: The first capture run

While attempting to feed the newly generated PCAP dump to the Joy tool, we realized that the BPF filter (mentioned earlier) was not being parsed correctly. The errors printed by Joy are shown in Figure 10.

```
Processing 1.pcap
error: could not parse filter ether host 08:12:A5:57:92:56: ethernet addresses
supported only on ethernet/FDDI/token ring/802.11/ATM LANE/Fibre Channel
error: could not parse filter ether host 00:55:DA:54:6D:B7: ethernet addresses
supported only on ethernet/FDDI/token ring/802.11/ATM LANE/Fibre Channel
error: could not parse filter ether host C8:2B:96:56:FA:5D: ethernet addresses
supported only on ethernet/FDDI/token ring/802.11/ATM LANE/Fibre Channel
```

Figure 10: Error messages printed by Joy

After a quick investigation, we realized that the link-type printed when we issued the first tcpdump command was LINUX_SLL, which was not the one we wanted. This was also confirmed by a Wireshark packet inspection of an arbitrary packet at the data link layer, as shown in Figure 11.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.1.1	192.168.1.222	SSH	120	Server: Encrypted packet (len=64)
2	0.000011	192.168.1.1	192.168.1.222	TCP	120	[TCP Retransmission] 22 → 53079 [PSH, ACK] S
3	0.000015	16:91:82:bb:03:90		LLC	120	I, N(R)=0, N(S)=32; DSAP ISO Network Layer (
4	0.000124	192.168.1.1	192.168.1.222	SSH	200	Server: Encrypted packet (len=144)
5	0.000129	192.168.1.1	192.168.1.222	TCP	200	[TCP Retransmission] 22 → 53079 [PSH, ACK] S
6	0.000131	16:91:82:bb:03:90		LLC	200	I, N(R)=0, N(S)=32; DSAP ISO Network Layer (
7	0.000183	192.168.1.222	192.168.1.1	TCP	66	53079 → 22 [ACK] Seq=1 Ack=65 Win=8207 Len=6

> Frame 12: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)

- ✓ Linux cooked capture v1
 - Packet type: Unicast to us (0)
 - Link-layer address type: Ethernet (1)
 - Link-layer address length: 6
 - Source: ASUSTekC_91:a3:00 (10:7b:44:91:a3:00)
 - Unused: 0000
 - Protocol: IPv4 (0x0800)
 - Padding: 000000

> Internet Protocol Version 4, Src: 192.168.1.222, Dst: 74.125.200.18

Figure 11: Arbitrary packet inspection of a Linux cooked capture.

After much research and testing, we realized that the interfaces were arranged such that the entire LAN can be captured from one interface displayed by `tcpdump`. This was discovered on the realization that a bridge labeled as `br-lan`, was connected to the entire Ethernet and WiFi collective, as displayed in the LuCI Interfaces menu pictured in Figure 12.

The screenshot shows the OpenWrt LuCI 'Interfaces' menu. At the top, there are two tabs: 'Interfaces' (active) and 'Global network options'. Below the tabs is a horizontal line, followed by the heading 'Interfaces'. Three interface cards are listed, each with a colored header and a list of statistics:

- LAN** (green header):
 - Protocol: Static address
 - Uptime: 2h 58m 10s
 - MAC: 16:91:82:BB:03:90
 - RX: 707.41 MB (516687 Pkts.)
 - TX: 1.59 GB (665443 Pkts.)
 - IPv4: 192.168.1.1/24
 - IPv6: fd48:9066:ded5::1/60
- WAN** (red header):
 - Protocol: DHCP client
 - Uptime: 2h 58m 9s
 - MAC: 14:91:82:BB:03:90
 - RX: 998.72 MB (545705 Pkts.)
 - TX: 700.06 MB (373054 Pkts.)
 - IPv4: 192.168.50.156/24
- WAN6** (red header):
 - Protocol: DHCPv6 client
 - MAC: 14:91:82:BB:03:90
 - RX: 998.72 MB (545705 Pkts.)
 - TX: 700.06 MB (373054 Pkts.)

Figure 12: List of interfaces displayed on OpenWrt LuCI.

The command `tcpdump -D` was issued to confirm the existence of `br-lan`, the results are printed on Figure 13.

```
root@OpenWrt:~# tcpdump -D
1.br-lan [Up, Running]
2.eth0 [Up, Running]
3.wlan0 [Up, Running]
4.eth1 [Up, Running]
5.eth0.1 [Up, Running]
6.wlan1 [Up, Running]
7.eth1.2 [Up, Running]
8.lo [Up, Running, Loopback]
9.any (Pseudo-device that captures on all interfaces) [Up, Running]
```

Figure 13: `br-lan` is available here, allowing us to capture all needed data.

Upon issuing the new `tcpdump` command (Figure 14).

```
root@OpenWrt:~# tcpdump -i br-lan -G 28800 -w /mnt/sdal/capture_%m-%d_%H-%M.pcap
tcpdump: listening on br-lan, link-type EN10MB (Ethernet), capture size 262144 byte s
```

Figure 14: The second capture run.

The link-type is exactly what it is supposed to be, Ethernet. The capture dump was retrieved from the router via SCP for inspection, and Wireshark displayed the correct format, as shown in Figure 15.

No.	Time	Source	Destination	Protocol	Length	Info
10	0.032824	192.168.1.125	3.227.250.255	TCP	66	56280 → 443 [ACK] Seq=1 Ack=298
11	0.032825	192.168.1.125	172.217.194.103	UDP	76	35177 → 443 Len=34
12	0.048856	3.227.250.255	192.168.1.125	TCP	66	443 → 56282 [ACK] Seq=1 Ack=1 W
13	0.048874	3.227.250.255	192.168.1.125	TCP	66	443 → 56282 [ACK] Seq=1 Ack=277
14	0.048888	3.227.250.255	192.168.1.125	TCP	66	443 → 56282 [ACK] Seq=1 Ack=555

> Frame 11: 76 bytes on wire (608 bits), 76 bytes captured (608 bits)

- ✓ Ethernet II, Src: f6:8c:78:20:78:63 (f6:8c:78:20:78:63), Dst: 16:91:82:bb:03:90 (16:91:82:bb:03:90)
 - > Destination: 16:91:82:bb:03:90 (16:91:82:bb:03:90)
 - > Source: f6:8c:78:20:78:63 (f6:8c:78:20:78:63)
 - Type: IPv4 (0x0800)
 - > Internet Protocol Version 4, Src: 192.168.1.125, Dst: 172.217.194.103
 - > User Datagram Protocol, Src Port: 35177, Dst Port: 443
 - > Data (34 bytes)

Figure 15: Arbitrary packet inspection showing the correctly captured packets.

Each output PCAP file was taken and renamed in a manner akin to the earlier data-set, the same shell script was used but the MAC addresses were replaced with the addresses of our own devices. The script was then executed to extract Joy JSON files. The PCAP dump illustrated above was fed into the joy shell script, the error did not appear and the JSON files were saved as expected, as pictured on Figure 16.

The data capture continued for a few days, all the devices were connected, and a variety of activities were done in a manner resembling the average household, to the best of the team's ability; Amazon Alexa commands were issued to ask random questions, Google Hub received the same treatment, several light control commands were sent via the cloud as well as in-LAN.

Before letting the capture run for a long time, it was important to verify that every packet was being properly captured, this was done by running the captures for an instance, and in that instance, several commands from different devices were issued simultaneously. The test capture file was then filtered down to each IP involved with the earlier actions, and the packet flows were observed for each device and their respective IP to see if the packets expected were captured, and they were.

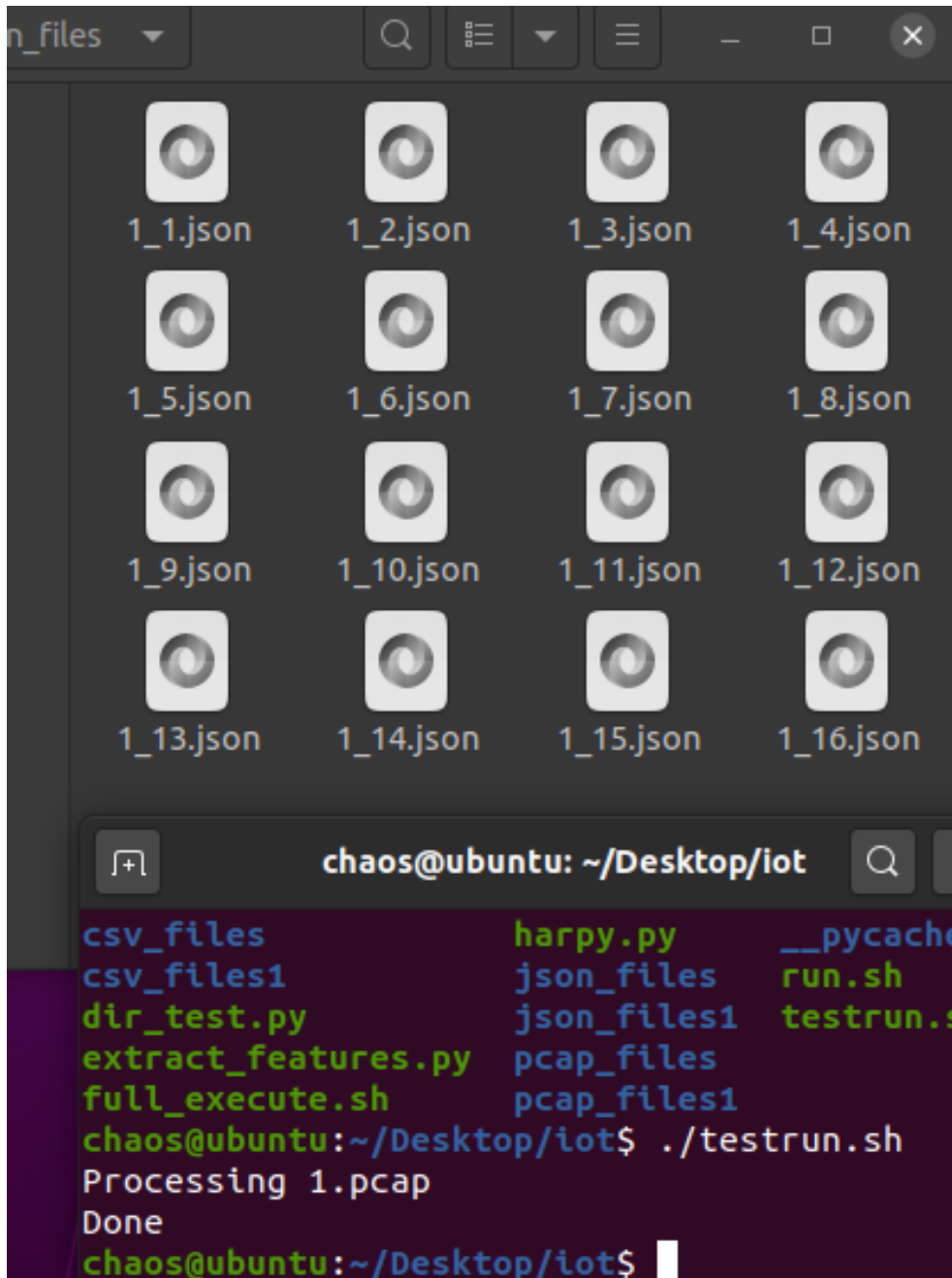


Figure 16: Successful Joy output.

At that point, the data collection was left to run on an even longer 1-day interval (-G 86400 seconds), just like the original UNSW data-set the team used in the

earlier stages of the project, with a minimal interruption involving the disconnection of the laptop where the Secure Shell was running. To prevent the recurrence of such an interruption and to maintain a smooth data collection process, `tmux` was used to execute the command (Figure 17), then shell window was detached to keep the process running after logging out of the SSH session, the 128GB space in the drive was sufficient, so long as the capture files are moved out to the processing PC on a regular basis and with the avoidance of downloads/uploads of massive media files like high-bitrate 8K footage, HDR content, uncompressed music files and next-generation high-budget video games, all of which could easily fill the drive if transmitted across the network.

```
root@OpenWrt:~# tcpdump -i br-lan -G 86400 -w /mnt/sdal/capture_%m-%d.pcap
tcpdump: listening on br-lan, link-type EN10MB (Ethernet), capture size 262144 bytes
[0] 0:tcpdump* "OpenWrt" 21:41 08-Nov-20
```

Figure 17: `tmux` for parallelizing workloads and keeping the capture process alive post logoff.

Comically speaking; the fact that a capture file was being transferred via SCP into the machine for processing caused `tcpdump` to copy the entire SCP transfer during an ongoing LAN capture, and once the copy was complete, the machine automatically uploaded the 10GB+ capture file into a NAS drive array set up on a different LAN, which resulted in the captured file in progress going from a few MBs to over 40GB in a few minutes. To avoid this, `tcpdump` was terminated, and the transfers were carried out immediately after the completion of the 24-hour interval by a team member on-site, causing slight continuous shifts in capture start times due to the post-capture transfer overhead.

7.4 Concerns and potential improvements

7.4.1 New Feature Overhead and Limitation

One of the milestones of the project was the integration of registry name fetching capability for use in the feature extraction function, in the earlier stages of development, it was observed that looking up RDAP info repeatedly added an insurmountable overhead which was deemed redundant during the rapidly successive runs. A solution was devised where all newly resolved names were dumped into a local JSON file for future executions. The caveat here is that RDAP records are not perfectly static but are often slow changing as many registries remain the same and rarely get updated, most registries are valid for at least one year and registrations are often renewed by the same entity.

A future update planned involves the addition of a `last_updated` key-value pair into the local RDAP records dump, the value would consist of string representing the date and time of the last time the local dump was updated. When the script runs and recognizes that the local dump is too old – the threshold can be manually defined here, this is up for future research – the script could proceed with wiping the record clean and re-build a new record, this is a more naïve approach which would wipe out correct and up-to-date record entries. A more intelligent approach would be to re-structure the data format to include the last update date for every single RDAP record entry, and re-query as necessary on an individual basis.

Another thing to keep in mind when using RDAP/WHOIS records as a feature for classification is the lack of universal applicability. The feature is not always viable in different regions as it was observed that many devices relied on some of the major CDN services like Amazon and Akamai where different servers with potentially different designated registry names may be used, it was observed when inspecting feature extraction output files that certain `asn_description` values carried additional strings with them, possibly linked to geographical location, as marked in Figure 18. CDNs are distributed across the world so that people (and devices) could easily and efficiently access content using the nearest available servers.

{'Not Resolved': 62, 'GOOGLE, US': 6, 'AKAMAI-AS, US': 17, 'AMAZON-02, US': 87}
{'Not Resolved': 354, 'AMAZON-02, US': 91, 'AKAMAI-AS, US': 115, 'GOOGLE, US': 29}
{'Not Resolved': 697, 'AKAMAI-AS, US': 53, 'GOOGLE, US': 98, 'AMAZON-02, US': 90, 'AKAMAI-ASN1, EU': 184}
{'Not Resolved': 374, 'GOOGLE, US': 1, 'AKAMAI-ASN1, EU': 126, 'AMAZON-02, US': 90, 'AKAMAI-AS, US': 50}
{'Not Resolved': 441, 'AKAMAI-AS, US': 95, 'AMAZON-02, US': 84, 'GOOGLE, US': 5}
{'Not Resolved': 47, 'AMAZON-02, US': 76, 'AKAMAI-AS, US': 12, 'GOOGLE, US': 5}
{'Not Resolved': 221, 'AMAZON-02, US': 82, 'AKAMAI-AS, US': 3, 'GOOGLE, US': 2}

Figure 18: Sample of a device’s extracted RDAP entry.

While the new RDAP feature does add some overhead to the preprocessing stage of the system, it is worth noting that capturing live traffic in the first place can be resource-taxing to the router responsible for mirroring traffic for the packet sniffer to store and process. The continuous capture of packets around the clock can be detrimental to the performance of the network. To address this issue, the packet sniffing tool can be triggered to run when certain unusual behaviors are detected from any device in the network, and only the traffic from that specific device can be sniffed in order to minimize the resources allocated by the router, as mentioned earlier, this is what has been referred to by one of the stakeholders as ”detect-on-exception”. Such unusual behavior could stem from non-concerning issues such as a sudden spike in activity attributed to rare actions such as device reconfiguration and

new user authentication, but can also be a signal to a malicious attack such as device hijacking. Additional features to collect can include the addition of ports and a list of protocols used, unusual protocols and ports can be an attribute of an anomalous behavior.

7.4.2 Router/Switch Requirements

Due to the potential overhead, it is recommended to run this system with a router that is capable of handling long data capture periods, with more on-board flash memory, RAM, and CPU power. The prototype used a Linksys WRT-1900ACS, which has a dual-core processor and 512MB of RAM, and day-long captures were performed with little compromise to the overall bandwidth of the router, but sustained transmissions caused by large file transfers can gradually throttle the bandwidth if the router was not capable of handling the traffic. The router must be configurable to allow for a sniffer to capture all the packets from the router's perspective, which grants the ability to see all packets going in and out of the LAN, as well as local transmissions. Not all routers can be configured in such a way using their Original Equipment Manufacturer (OEM/stock) firmware, and in some cases, flashing them with alternative firmware may be the only option. This is particularly true for newer, niche-marketed routers such as the ASUS ROG AX11000 and AXE11000, where the target consumer is not expected to tinker with such settings in the first place (in the case of those routers, the answer is Asuswrt-Merlin). On the other hand, enterprise-grade managed switches are aimed for professional IT administrators and network engineers, where the deepest level of access and the most fine degrees of control over a switch's configuration settings is expected, if not outright required. Therefore not warranting the replacement of the OEM firmware in favor of something like OpenWrt.

7.4.3 Data-set-related Limitations

Feature extractions were done using a select set of intervals; 5 minutes, 15 minutes, and 30 minutes. This was done to see how devices behave in different time windows. The details would be discussed later in the report, but there is an argument to be made; in certain real cases, some devices may behave in a way that could be deemed absurd and anomalous when inspected within the confines of a limited time interval, without context whatsoever. Meaning that in some cases, a short fixed interval would in no way be representative of a device's usual overall behavior. Case at hand; NAS drives continuously pump out files in a media production studio throughout business days, it is the norm in high-budget productions that each post-production unit is connected to a major near-PB sized NAS drive connected by 10GbE and transmit Gigabytes of data per second, continuously, for up to multiple hours. REDCODE 8K 12:1, 30FPS, HDRX-enabled Raw video footage is encoded at around 270MB/s, multiple separate files for different projects would boost this to GB/s if multiple production units were in place, more info here: <https://www.red.com/recording-time> Such enormous streams of data may be the norm in some environments, while in the context of the average household, this may be unusual. Hence why context is very important, and accounting for all sorts of usage scenarios would most definitely come in handy.

Lastly, the IoT network that the team set up is of a small scale but has allowed us to collect live data for our own research. A much bigger IoT network with more device variety would help in generating more robust data-sets that would lead to achieving better classification results for more devices, taking account all the different kinds of IoT devices demonstrating all sorts of different network behavior.

More users interacting with the IoT network would also grow the data-set to account for different usage behaviors by different users. People that are home most of the time would interact with the IoT devices differently compared to others who are mostly absent and rarely ever interact with said devices. The situation is not ideal, the team's budget was limited and only a few devices were procured for testing, and only a single person was interacting with all the devices regularly.

We see companies inventing all kinds of new IoT-capable devices regularly and it is imperative that we be able to adapt to the rapidly changing market by improving the detection system to account for any new types of devices demonstrating never-seen before functions that may break into new classification categories.

8 Machine Learning Software Program

8.1 Introduction

This section of the report will be a walk-through of the software part component of the project. This part of the project includes all the elements that perform the process of studying the traffic data collected and prepared, then running it against a machine learning module that will determine the type of the device based on the traffic behavior. After this is processed and the device type is determined, the program, in turn, will update the real-time database on Firebase, thus the detection will also be shown on the interface (App) in real-time.

We will split this section of the report into multiple sub-sections, each will contain a description of one of the building blocks of our machine learning software program.

8.2 Machine Learning Program Functionality

The main functionality of our machine learning program is to study the traffic captured through the router, and predict the device type based on it. Each device connected to the network will have a unique network behavior that makes predicting its type or category possible. The device network behavior is determined through a set of features that we chose after thorough research. What the machine learning model essentially does is that it compares the network traffic packet associated with a specific device with the previously studied packets while training the model, based on that, the model predicts the device type.

Essentially, the goal is to reach the highest prediction accuracy possible. The accuracy of the machine learning program depends on a variety of factors, such as the size of the training data-sets, the prediction features, and the correctness of the input packet. In this project, we tried to maximize the efficiency and accuracy of our machine learning model by taking all these factors into consideration.

In an attempt to give the user more options and provide them with more flexibility, we decided to give the user the ability to choose from one of the three time-scales for capturing. However, as a consequence, there is a trade-off between the speed of capturing and the accuracy of prediction, i.e. the longer the capture, the higher the accuracy would be. During implementation, we created three machine learning models, each tailored to one of the time-scales chosen.

This component of the project goes through a few main stages which are the following:

1. Prepare the data-sets for training each machine learning model.
2. Train the models.
3. Prepare the testing data.
4. Collect the testing results.
5. Listen to captured traffic.
6. Extract the time scale chosen by the user from the database.
7. Choose the model to use and feed the captured file to it.

8. Go over the captured data in the sheet and predict the devices with the ML model.
9. Extract the device information.
10. Update the Firebase database with the predicted devices.

We will describe these stages in further detail in the following sub-sections.

8.3 Machine Learning Algorithm

There has never been a better time to get into machine learning. With all the resources available online, machine learning became a field that is democratized by the internet. There are many machine learning algorithms available, such as; Linear Regression, Decision Trees, and Neural Networks. However, in this project, we decided to use the Random Forest algorithm to predict the device type and perform classification.

Random Forest is an easy to use and flexible machine learning algorithm that produces, even without hyper-parameter tuning, great results most of the time. Because of its simplicity and diversity, it is one of the most used algorithms.

The way Random Forest works is that it simply builds multiple decision trees and merges them together to get a more accurate and stable prediction. Random Forest has nearly the same parameters as a decision tree, but it enhances the performance by merging multiple decision trees together, the figure below visualizes the functionality of the Random Forest algorithm:

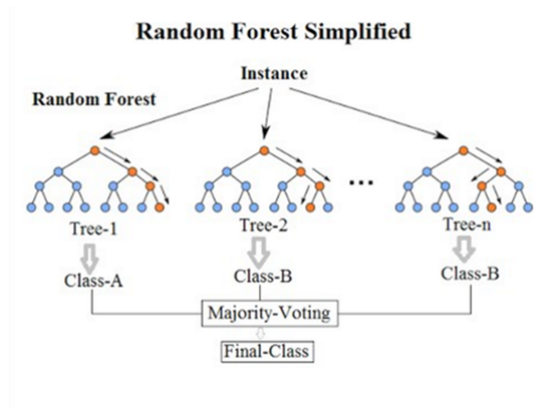


Figure 19: Random Forest algorithm functionality simplified

Adding to the advantages we mentioned above, we also chose Random Forest because it adds some additional randomness to the machine learning model when growing trees. Therefore, instead of searching for the most important feature while splitting a node, it searches for the best feature among a randomly selected subset of features. This leads to a wide diversity that results in a better machine learning model.

8.4 Data Preparation

The first step in writing a machine learning program is preparing and cleaning the data files for training. Data preparation is needed because the data received from the data acquisition stage might contain some errors or deficiencies that need to be fixed before feeding the files to the machine learning model. As an example, errors can be caused by rows that have missing pieces of data. Moreover, some of the features included in the data files are only needed for the data acquisition stage, so these features need to be removed as they might affect the accuracy of the model. The figure below shows a snapshot of the training data-set that we used in this project.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	total_sleep	total_active	total_flow_v	flow_rate	avg_packet	num_servers	num_protoc	uniq_dns	dns_interval	ntp_interval	device_co	Device Name	MAC_address	Connection Type	Device Type
2	42	279	1763	6.31899642	22.6025641	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
3	44	244	1114	4.56557377	17.9677419	2	3	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
4	23	312	1518	4.86538462	18.975	2	3	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
5	65	262	1631	6.22519084	23.3	2	3	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
6	34	265	992	3.74339623	15.2615385	2	1	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
7	80	269	1893	7.03717472	25.9315068	2	3	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
8	34	285	1128	5.95789474	16.8358209	2	2	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
9	74	317	1742	5.49526814	22.3333333	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
10	39	289	1286	4.44982699	16.9210526	2	2	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
11	26	277	1906	6.88086643	25.4133333	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
12	35	268	938	3.5	15.3770492	2	1	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
13	59	280	1672	5.97142857	23.2222222	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
14	44	305	1808	5.92786885	24.7671233	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
15	40	267	1077	4.03370787	15.8382353	2	1	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
16	28	282	1755	6.22340426	24.7183099	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
17	46	285	1408	4.94035088	18.0512821	2	2	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
18	68	296	2049	6.9222973	25.2962963	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
19	35	260	1130	4.34615385	16.8656716	2	2	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
20	39	300	2033	6.77666667	24.7926829	2	4	1	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub
21	31	291	993	3.41237113	14.6029412	2	2	0	0	0	0	1 Smart Things	d0:52:a8:00:67:5e	Wired	Hub

Figure 20: Snapshot of training data-set

We created a tailored data-set containing around 800,000 rows of data for each of the three machine learning models. We obtained the data-sets from the The University of New South Wales in Sydney. They have a public website [12] containing a data-set originating from the previous research efforts on IoT device classification, consisting of 20 days worth of captures for over 20 devices. When running a python piece of code to calculate the number of devices contained in one of the files according to their type, this is the output we got:

```

Camera      21858
Hub         17331
Sensor      12052
Misc        11225
Appliance  10429
Computer    3376
Name: Device Type, dtype: int64
Number of types detected is: 6 types

```

Figure 21: Data-set file analysis

The device types or categories were chosen by us. We tried to be as general as possible and choose classification types that cover most of the IoT devices that we might capture.

Each of the data-sets contain a set of features that will be used to train the model. The features we used to train the model are the following:

1. Total Sleep Time (Seconds). It indicates the time duration the device was not active on the network.
2. Total Active Time (Seconds). It indicates the time duration the device was active on the network.
3. Total Flow Volume (Bytes). It indicates the volume of the traffic flow from the device into the network.
4. Flow Rate (Bytes/Second). It indicates the rate of the data flow from the device into the network.
5. Average Packet Size (Bytes).
6. Number of Servers. It indicates the number of servers the device is connected to.
7. Number of Protocols. It indicates the number of network protocols used by the device.
8. Unique DNS. It indicates the unique domain of the device.
9. DNS Interval.
10. NTP Interval

11. Connection Type. It indicates whether the device is wired or wireless.
12. RDAP/WHOIS Record. It shows the number of connections associated with the registration name associated with an IP.

These are the main features that are going to be used to train the model as independent variables, which the model will use to predict the classification type of the device. We conducted a thorough research to determine the most useful and unique features in order to get the highest possible accuracy.

It is also important to mention that a new 12th feature was introduced at a later stage in the project which was indicated previously in the data processing section of the report, which is the RDAP/WHOIS feature. After adding this feature we noticed an increase in the accuracy of the machine learning model.

On the other hand, we have only one dependent variable in the training data-set which is the device type. There are also some of features that are unnecessary such as the `device_co` which was used in the data acquisition stage. Also, we avoided using features like the device name or the MAC address because they can lead to errors and would affect the accuracy of the model.

For the data preparation process we used the Python library `pandas` that is “Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the Numpy package and its key data structure is called the DataFrame.” [13] The captured files of the network traffic are fed to the machine learning model in a `.csv` format. The first step is to open and read the files, and we do it with the following python commands:

```
# Training set for the 5 min time scale
df_5m_unsw = pd.read_csv("unsw_dataset_features_mapped_5m.csv", low_memory=False)
# Training set for the 15 min time scale
df_15m_unsw = pd.read_csv("unsw_dataset_features_mapped_15m.csv", low_memory=False)
# Training set for the 30 min time scale
df_30m_unsw = pd.read_csv("unsw_dataset_features_mapped_30m.csv", low_memory=False)
```

Figure 22: Open the CSV files

After loading the files, we start by handling the missing values to remove faulty rows. This is an important step to assure the correctness of the machine learning model training. What we basically do is drop any row of data that is missing values. We achieve that using the following commands:

```
# handle missing values
df_5m = df_5m.dropna()
df_15m = df_15m.dropna()
df_30m = df_30m.dropna()
```

Figure 23: Drop any row of data with missing values

We also need to drop the unnecessary features in each data-set which we mentioned above, and we achieve that using the following command:

```
# remove the unnecessary data columns
df_5m.drop(["device_co", "Device Name", "MAC_address"], axis=1, inplace=True)
df_15m.drop(["device_co", "Device Name", "MAC_address"], axis=1, inplace=True)
df_30m.drop(["device_co", "Device Name", "MAC_address"], axis=1, inplace=True)
```

Figure 24: Drop the unnecessary features

The next step is to encode any non-integer features to allow the model to process it. For this specific function we used the One-Hot encoder from the python library called `sklearn`, which we later on used to create the Random Forest machine learning model. Essentially, the One-Hot encoder represents categorical variables as binary vectors. At first the categorical values are mapped to integer values. Then, each of the integer values is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1. After importing the `OneHotEncoder` from the `sklearn.preprocessing` library, we used the following commands to encode the categorical values in the data-sets:

```
# convert strings into numbers so the model can process it
le = preprocessing.LabelEncoder() # use LabelEncoder
df_5m['rdap_asn'] = le.fit_transform(df_5m['rdap_asn'])
df_5m['Connection Type'] = le.fit_transform(df_5m['Connection Type'])
df_15m['rdap_asn'] = le.fit_transform(df_15m['rdap_asn'])
df_15m['Connection Type'] = le.fit_transform(df_15m['Connection Type'])
df_30m['rdap_asn'] = le.fit_transform(df_30m['rdap_asn'])
df_30m['Connection Type'] = le.fit_transform(df_30m['Connection Type'])
```

Figure 25: Encode the categorical values

In our case, we only have to encode the RDAP and connection type features, as they are the only non-numerical features in our training data-sets.

The last step in the data preparation process is to define the dependent and independent variables. We have only one dependent variable which is the device type, it is basically the target that we want the machine learning model to predict. The rest of the features are used as the independent variables, and they are the 12 features we previously listed. We split the features into X data-set (representing the independent variables) and Y data-set (representing the dependent variable). To achieve this we run the following command to create the Y data-set:

```
# define the dependent variable
Y5m = df_5m["Device Type"].values
Y15m = df_15m["Device Type"].values
Y30m = df_30m["Device Type"].values
```

Figure 26: Create the Y data-set

And the following commands to create the X data-set:

```
# define the independent variable
X5m = df_5m.drop(labels=["Device Type"], axis=1)
X15m = df_15m.drop(labels=["Device Type"], axis=1)
X30m = df_30m.drop(labels=["Device Type"], axis=1)
```

Figure 27: Create the X data-set

Where we basically store all the features except for the device type by dropping it.

8.5 Machine Learning Model Creation and Training

As we mentioned above in this section, we decided to use the Random Forest algorithm and we explained the reasons behind our choice, and how the algorithm works. Now, we will go over the process we used to create and set up the three machine learning models in our projects. As we need three models, one for each of the three time scales, we create three Random Forest models and train each of them with the respective network traffic data-sets that we prepared.

For this process we use the Python library `sklearn`. Sci-Kit Learn (`sklearn`) is a Python module integrating a wide range of state-of-the-art machine learning algorithms for medium scale supervised and unsupervised problems. This package focuses on bringing machine learning to non-specialists using a general-purpose high-level language. Emphasis is put on ease of use, performance, documentation, and API consistency [14].

The first thing we do is import `train_test_split` from the `sklearn` library, we use it to split the data-sets into two parts, one for training and one for testing, then we store each inside a dedicated variable to use in the model creation command. We also had to indicate the size of the testing part, in our case, we decided to make it 20% of the data-set size. A `shuffle` attribute was also selected to be true to help achieve maximum accuracy. To achieve this, the following commands were used:

The next step was to create and train the model. First we import `RandomForestClassifier` from `sklearn.ensemble` and then use it to create our three machine learning models. To do so we used the following commands:

```

# split data into training set and testing set
from sklearn.model_selection import train_test_split
# 5 minutes time scale model
X_train_5m, X_test_5m, Y_train_5m, Y_test_5m = train_test_split(X5m, Y5m,
                                                                test_size=0.2, random_state=10, shuffle=True)
# 15 minutes time scale model
X_train_15m, X_test_15m, Y_train_15m, Y_test_15m = train_test_split(X15m, Y15m,
                                                                      test_size=0.2, random_state=10, shuffle=True)
# 30 minutes time scale model
X_train_30m, X_test_30m, Y_train_30m, Y_test_30m = train_test_split(X30m, Y30m,
                                                                      test_size=0.2, random_state=10, shuffle=True)

```

Figure 28: Split the data-sets for training and testing

```

# create the model
from sklearn.ensemble import RandomForestClassifier

model_5m = RandomForestClassifier(n_estimators = 10, random_state=30)
model_15m = RandomForestClassifier(n_estimators = 10, random_state=30)
model_30m = RandomForestClassifier(n_estimators = 10, random_state=30)

```

Figure 29: Import RandomForestClassifier from sklearn

After initializing the models, we train them using their corresponding training data-sets, which contain the features we captured in the data acquisition stage, then cleaned and prepared in the data preparation stage. To train the models, the following commands were used:

```

# train the model
model_5m.fit(X_train_5m, Y_train_5m)
model_15m.fit(X_train_15m, Y_train_15m)
model_30m.fit(X_train_30m, Y_train_30m)

```

Figure 30: Train the model

Now the machine learning models are trained, setup, and ready to be used for classification and prediction.

8.6 Testing and Results Observations

After we trained the machine learning model, we tested it to see the accuracy of prediction and the importance of each feature. To run a test on the model with the testing data-sets that we created, we used the `.predict()` method as shown in the code below:

```
# test the model
prediction_test_5m = model_5m.predict(X_test_5m)
prediction_test_15m = model_15m.predict(X_test_15m)
prediction_test_30m = model_30m.predict(X_test_30m)
```

Figure 31: Test the model

Then we imported a sub-library called `metrics` from `sklearn` to analyze the testing results and acquire the importance of each feature. We used the following commands to find the results:

```
print("Testing results for the 5 minutes time scale model...")
print("Accuracy = ", metrics.accuracy_score(Y_test_5m, prediction_test_5m))
# check the contribution of each independent variable
feature_list = list(X5m.columns)
feature_imp = pd.Series(model_5m.feature_importances_, index=feature_list).sort_values(ascending=False)
print(feature_imp)
```

Figure 32: Find the results

We repeated the same process for each of the three time scales to observe the differences. The following figures show the Python output that shows the performance stats of each machine learning model:

```
Testing results for the 5 minutes time scale model...
Accuracy = 0.9850321808112558
avg_packet_size      0.164315
total_flow_volume    0.148736
Connection Type      0.129154
num_servers           0.114969
uniq_dns              0.104679
num_protocols         0.090511
flow_rate             0.079706
total_sleep_time     0.068104
total_active_time    0.058445
rdap_asn              0.036782
ntp_interval          0.004514
dns_interval          0.000085
dtype: float64
```

Figure 33: Performance analysis for the 5 minutes time scale model

```
Testing results for the 15 minutes time scale model...
Accuracy = 0.987363100252738
avg_packet_size      0.194442
total_flow_volume    0.137380
num_protocols         0.118807
num_servers           0.116731
total_sleep_time     0.101997
Connection Type      0.087000
flow_rate             0.079359
uniq_dns              0.071524
total_active_time    0.042310
rdap_asn              0.041121
ntp_interval          0.009229
dns_interval          0.000101
dtype: float64
```

Figure 34: Performance analysis for the 15 minutes time scale model

As we predicted, we observed that the accuracy increases when the time scale increases, and this happens because the bigger the time scale, the more data will be captured, hence the more accurate the predictions will be.

```
Testing results for the 30 minutes time scale model...
Accuracy = 0.9886947584789312
avg_packet_size      0.207244
num_protocols        0.133521
total_flow_volume    0.130173
uniq_dns             0.113721
num_servers          0.106783
flow_rate            0.088294
total_sleep_time     0.085706
total_active_time    0.054557
rdap_asn             0.054032
Connection Type      0.015079
ntp_interval         0.010779
dns_interval         0.000112
dtype: float64
```

Figure 35: Performance analysis for the 30 minutes time scale model

However, it is important to mention here that there is a key trade-off for the user, where the user has to choose between speed and accuracy. If the user chooses a bigger time scale, the results will be more accurate, but the detection of new devices will be slower. On the contrary, if the user chooses a smaller time scale, the results will be less accurate, but the detection will be faster.

The testing accuracy results might seem quite high and close, but we believe that was due to the fact that we did not use a huge variety of devices, where our training and testing data-sets contained around 20 devices. However, we tried our best to include as many devices variations as possible. We believe that the results will have the same behavior that shows the direct correlation between the time scale and the accuracy of the model, but the accuracy will definitely be lower if we use bigger data-sets.

Another key observation was the importance of each feature in the prediction process. The average importance of each feature is illustrated in the chart below:

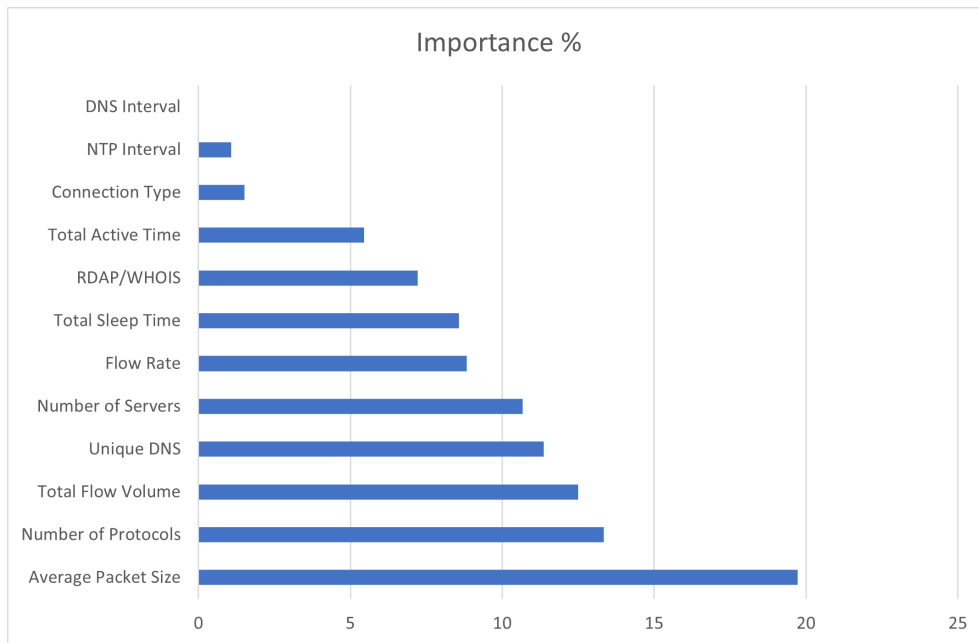


Figure 36: Importance of each feature

8.7 Listening and Updating the Real-time Database

The final step in this section of the project is what we think of as the engine of Harpy. In this part, we will make the program listen to the captured data that will be fed to the machine learning program through the data acquisition program. The data acquisition program will update the CSV file that will be stored in the SBC (Tinker Board) which will be running the machine learning model. Based on the changes, the machine learning program will add any newly detected devices to the Firebase real-time database, which in turn will automatically update the mobile phone application real-time as well.

The first thing we needed to do is to configure the Firebase real-time database. For this we used the Python library pyrebase. To achieve this, we used the following commands:

```
# firebase configuration
config = {
    "apiKey": "AIzaSyBJqqj1jgw0sNgCD-iC_uKbKPLqdg5Ytp8",
    "authDomain": "harpy-c8519.firebaseio.com",
    "databaseURL": "https://harpy-c8519.firebaseio.com/",
    "projectId": "harpy-c8519",
    "storageBucket": "harpy-c8519.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()
ID = '-MHfQDfcv017vmKEFU0c'

timeScale = db.child(ID).get().val()['timeScale']
```

Figure 37: Firebase Configuration

Through this piece of code, we connect the machine learning model to the Firebase real-time database and obtain the time scale selected by the user in the application. Based on the time scale value obtained from the database, we choose the corresponding machine learning model to use for prediction.

After we configure the Firebase database, we prepare the captured file in the same way we prepared the training data-sets. Finally, we run a continuous program that listens to new devices, and updates the real-time database accordingly. This program will run based on the time scale chosen by the user, so if the user selects the 5-minute time scale, the program will run every 5 minutes. This is achieved using the following piece of code:

```
while (1):
    captured_set_sample = pd.read_csv("test_output.csv") # A sample of a captured dataset
    captured_set_sample = captured_set_sample.dropna()
    captured_set_sample_enc = captured_set_sample[0:]

    le = preprocessing.LabelEncoder() # use LabelEncoder
    captured_set_sample_enc['rdap_asn'] = le.fit_transform(captured_set_sample_enc['rdap_asn'])
    captured_set_sample_enc['Connection Type'] = le.fit_transform(captured_set_sample_enc['Connection Type'])
    captured_set_sample_enc.drop(["Device_co", "Device Name", "MAC_address", "Device Type"], axis=1, inplace=True)

    db_content = []
    timeScale = db.child(ID).get().val()['timeScale']
    if (timeScale == 5):
        chosenModel = model_5m
    elif (timeScale == 15):
        chosenModel = model_15m
    elif (timeScale == 30):
        chosenModel = model_30m

    for row1, row2 in zip(captured_set_sample_enc.iterrows(), captured_set_sample.iterrows()):
        features = list(row2)[1]
        if (not first_row):
            if (features[13].strip() not in db_content):
                device_type = chosenModel.predict(list(row1)[1:12])

                db_content+= [features[13].strip()]
                if(device_type[0]=='Hub'):
                    iconName = 'device-hub'

                elif(device_type[0]=='Camera'):
                    iconName = 'camera-alt'

                elif(device_type[0]=='Computer'):
                    iconName = 'computer'

                elif(device_type[0]=='Appliance'):
                    iconName = 'home-filled'
```

Figure 38: Listen and update - 1

```

elif(device_type[0]=='Sensor'):
    iconName = 'settings-remote'

elif(device_type[0]=='Misc'):
    iconName = 'miscellaneous-services'

elif(device_type[0]=='Network'):
    iconName = 'router-wireless'

db.child(ID).child(features[13].strip()).update({
    "Device Type": device_type[0],
    "Device MAC": features[13].strip(),
    "Device Name": features[12].strip(),
    "Device Total Active Time": features[1],
    "Device Total Flow Volume": features[2],
    "Device Flow Rate": features[3],

```

Figure 39: Listen and update - 2

```

    "Device Avarage Packet Size": features[4],
    "Number of Servers": features[5],
    "Number of Protocols Used": features[6],
    "iconName": iconName,
    "safety": False,
})
first_row= False
sleep(timeScale*60)

```

Figure 40: Listen and update - 3

9 Output System

9.1 Overview

After we process the captured traffic data, analyze it and classify the devices, we need to give the user the output in the most friendly way possible. Two methods will be used to alert the user and allow them to interact with Harpy.

The first method is the hardware method, where a small hardware device (the SBC, as mentioned earlier), that has the shape of a puck, will be placed in the vicinity. This device will alert the user by lighting red if Harpy detected an anomalous deployment of any IoT device. The hardware device will consist of a Raspberry Pi, with an LED attached to it. We plan to use a WiFi equipped Raspberry Pi in order to connect it to the real-time database that we will talk about later on. The Raspberry Pi will flash a red light in the case of an alert.

On the other hand, our second method is a mobile phone application. As we thought that the user might want to get notified about an alert even if they were outside the premises, which in turn would help our users take immediate action and stay updated anytime and anywhere. The application will consist of the major parts as shown in the appendix. A sign-in page, to provide security for the user. A dashboard page, where the user can view details about their devices and view the recent activities. And a devices management page where the user can manage by adding or removing their personal IoT devices that they want the program to recognize. The app will be developed using React Native and it will function on both platforms, iOS and Android. In the case of an alert, the user will receive a notification on their mobile device through the application.

9.2 React Native

React-Native is a JavaScript (JS) framework for developing native mobile applications in both iOS and Android. The term React-Native was derived from React which is a JavaScript library, and native which is a mobile application type [8]. React was created by Facebook in 2013 that helps build user interfaces for web applications. It was well received by the public, making it the 5th most starred repository on GitHub as it allows for modular and reusable coding by utilizing a component structure [8]. A native app means that the app is developed for a particular operating system such as iOS or Android. To code such apps, one can use Swift for example to build an iOS native app. And to build a native app for Android, Java can be used.

However, the beauty of React Native is that it allows you to build one native app, and write one piece of code, but then it compiles the .js file into two executables; one for iOS and one for Android. This is what makes React Native very special and it explains the growing use of this programming language. That is the main reason we chose to build our app using React Native. Native mobile apps have access to the device's APIs and features making them high-performing and reliable [8]. Therefore, they can provide optimized performance and can take advantage of the latest technology within the mobile phones and tablets compared to non-native apps. Native apps installed directly on a mobile device and does not make use of WebView which ranks it higher than non-native ones in terms of performance. Although it feels and looks like a real native application, React-Native is coded by simply using React and JavaScript. Real native apps have to be coded in two different languages, one for iOS and another for Android. They always outperform apps coded in a non-native fashion, but they require more time, people, and financial resources, therefore, developers opt for React-Native which is cross-platform but also native in a way [8].

As for our project, we plan to use React Native to build a mobile app that connects to an online real-time database that we will create using a backend service. This database will be the bridge that connects our machine learning module with our output system. React Native works in our favour here as it has many built-in libraries that support backend services like Firebase and AWS. We have explored React Native and built a skeleton code for our three main components of the app, as shown in the appendix below.

9.3 AWS

AWS includes many services for IoT devices, databases, machine learning and analytics. The database service called AWS Simple Storage Service or AWS S3 is a large cloud-based database system where users can store any kind of files and then access them anytime and anywhere. Big companies such as Netflix and AirBnB use this service to back up content and files that are served to the users of the services. The service uses the concept of buckets and tags to categorize stored objects. This makes it easier to update the objects when using the S3 batch operations, which allows objects to be managed and moved in groups [9]. AWS Lambda functions can be triggered to manage specific parameters among the objects.

Services related to Internet of Things device management and controls are also provided by AWS IoT Events, IoT Core and IoT Device Management. IoT Events allows easy detection of events from devices and sensors, analysis of the data collected and response to those events [10]. Collecting data from devices and acting on that data enables more connections between them and makes the devices smarter and better automated. Using IoT Core and Device Management, an inventory of IoT devices can be created, managed and organized. IoT Core is a cloud based platform that uses MQTT and Websockets protocols to allow for a publisher/subscriber connection between it and the devices [11]. From such a structure, multiple devices can have their data published to the cloud and accessed by subscribers that can use that data for analysis or decision making processes. Information about the temperature, energy consumption, functionality, time used and user patterns can be useful from IoT devices in order to create better applications that will make the use of such devices more efficient to the user.

All the features and services mentioned above are very interesting and can serve many purposes. However, they are more focused on the performance of the native app rather than the speed and security of the app's functionality. They perform powerful analytics and operations, but they all come with big costs.

9.4 Firebase

Firebase is a mobile and web development platform developed by Firebase Inc. and then acquired by Google in 2014. They are considered the direct competitors of AWS. Firebase offers a wide range of backend services, from analytics to cloud messaging. However, for our project, the main services that concern us are the Firebase Authentication service and the Firebase Realtime Database service. These two services are extremely crucial in the development of our mobile application as they will allow us to assure user privacy and create the bridge between our machine learning, the hardware component and our software output component.

Firebase Authentication is a service that can authenticate users using client-side code. This service allows the developer to build an efficient and secure login system. It functions in a similar manner to AWS's user pool. However, it focuses more on security than quantity, as it offers a vast variety of authentication and login options, such as logging in with email or even authenticating the user through SMS. It provides very powerful APIs to perform such operations and implement such features. It is also pretty easy to integrate into a React Native application as shown in the appendix section below. Moreover, it supports social login providers like Facebook and Google. Additionally, it includes a user management system whereby developers

can enable user authentication with email and password login stored with Firebase. As our project's main goal is to provide security for the user, it only makes sense that we build the application with a solid security login system, in which this service will come in handy.

Firebase provides a real-time database and back-end as another service. The service provides application developers an API that allows application data to be synchronized across clients and stored in Firebase's cloud. The developer here can start using the real-time database which can secure their data by using their own server-side-enforced security rules. As for our project, we need a gateway to connect the machine learning program to the mobile app that will show the output. Using Firebase's real-time database, the ML module will update it every time a new IoT device is detected. On the other side, the application will be connected to the same real-time database as shown in the appendix section below. It will basically check the database frequently and reflect the output to the user. In the case of an alert, the application will send a notification to the user through the app.

9.5 Firebase vs AWS

AWS has more options for features and services that they provide. However, they are mostly out of the scope of our project. It is also expensive and more difficult to set up than Firebase. On the other hand, Firebase is very easy to set up (as shown in the appendix section below) and more related to this project. With Firebase Authentication and Firebase Real-time Database services, the process of building our mobile application and connecting it to our machine learning module will become much easier and more efficient. That is why we opted to use Firebase in this project.

9.6 The Single Board Computer

In this project we need a device that can capture data from the network and run the machine learning algorithm to determine the types of the devices, and finally update the database in real-time. In the first phase of the project we have used Raspberry Pi 3 to do all that. It was very easy to use with Raspbian operating system. We faced no issues configuring the board and installing the libraries we needed. Also, we connected to the Raspberry Pi using Remote Desktop Connection (RDC) on Windows. The only issue with the Raspberry Pi is that it was sluggish and not powerful enough to do all these tasks.

We did our research and we found some of the best Single Board Computers (SBCs) and one of them caught our eyes. That was the ASUS Tinker Edge R. This SBC is specifically designed for AI applications that uses Rockchip NPU (Neural Processing Unit), a machine learning accelerator with low power consumption. The board has hex-core ARM Cortex processor, a 4 GB dual-channel LPDDR4 memory, and a 2 GB of standalone memory for the NPU. We acquired the Tinker Edge R SBC. We had some issues in the beginning with flashing the OS and configuring the board, but now it is up and running. The board performance is astonishing. We copied the Python scripts—that captures the network traffic, processes the captured data, and updates the database—to the board and we did not face any slowness or overheat from the board at all.

The system works as follows. The board is installed and configured initially with the safe devices in the perimeter and is connected to the database with a unique key that we provide. The board will be always on and capturing data. The machine

learning module will be triggered only by the mobile app when the user uses it. Depending on the time scale of the captured data that the user chooses, the ML module will then start processing the data, and acquire the list of devices that are in the perimeter, then will update the list on the mobile app real-time.

9.7 The Mobile Application

We built the mobile app using React Native as we previously elaborated. To begin with, we tried to make the mobile app as simple as possible for the best user experience. The app is made of 3 pages. The home page, or the landing page, is what the user first sees after opening the app. The second page is where we show the list of devices that we fetch from the database. Finally, the third page is where we show the device details. Regarding the design; we tried to have a simple sharp theme to reflect that the app is related to new tech.

For the main page there is the logo that we got from hatchful (<https://hatchful.shopify.com>) alongside with simple instructions. There are 3 buttons to give the user the choice to select whichever time scale he wants for the captured data, either 5, 15, or 30 minutes. Then comes the scan button, that when pressed, triggers the ML module on the SBC to process the data that was previously captured and then updates the database, then it navigates to the next page, the devices list page, and shows all the devices.

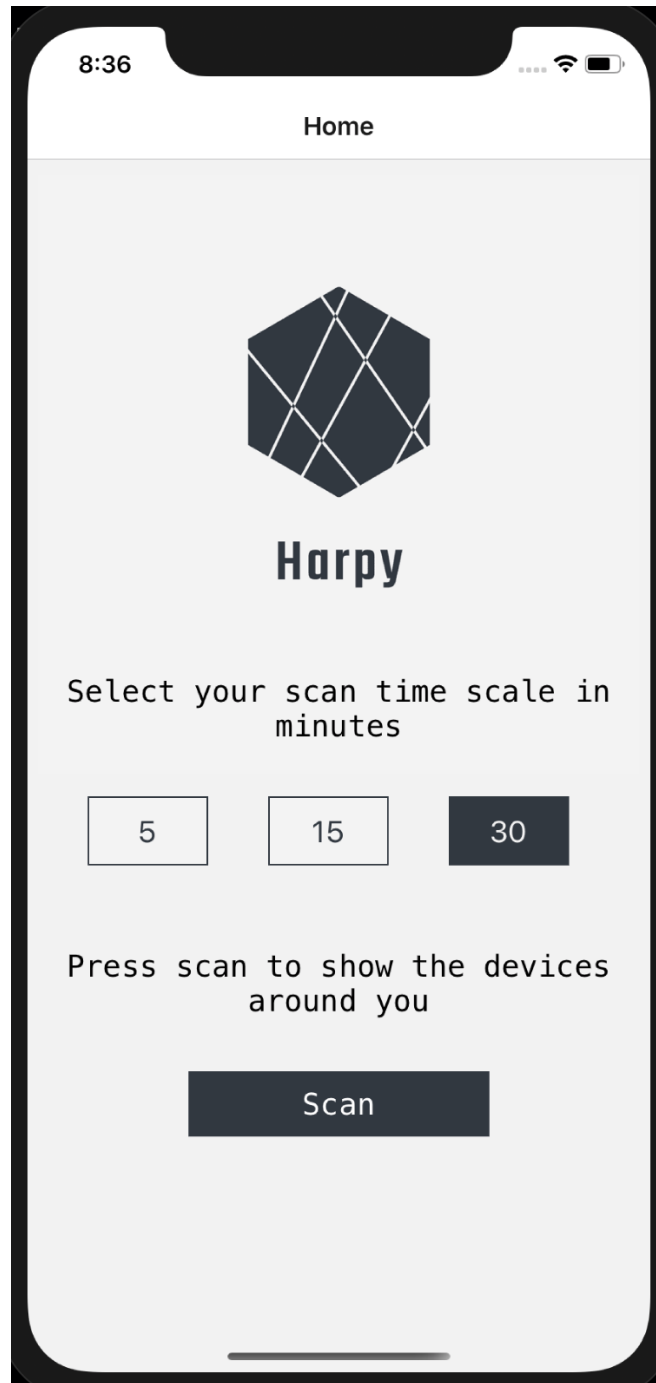


Figure 41: Mobile app first page

The second page shows all the devices that were acquired from the SBC and submitted to the database. We fetch the devices, and we show them on the screen using Flat List. We use an icon that represents the device, that can also be green or red depending on whether the device is safe or not, which is determined by the user and saved on the database. We show some concise details about each device such as the device type and its MAC address. The list is made of buttons that the user can scroll over and can click to go to the third page that shows more details about the device.

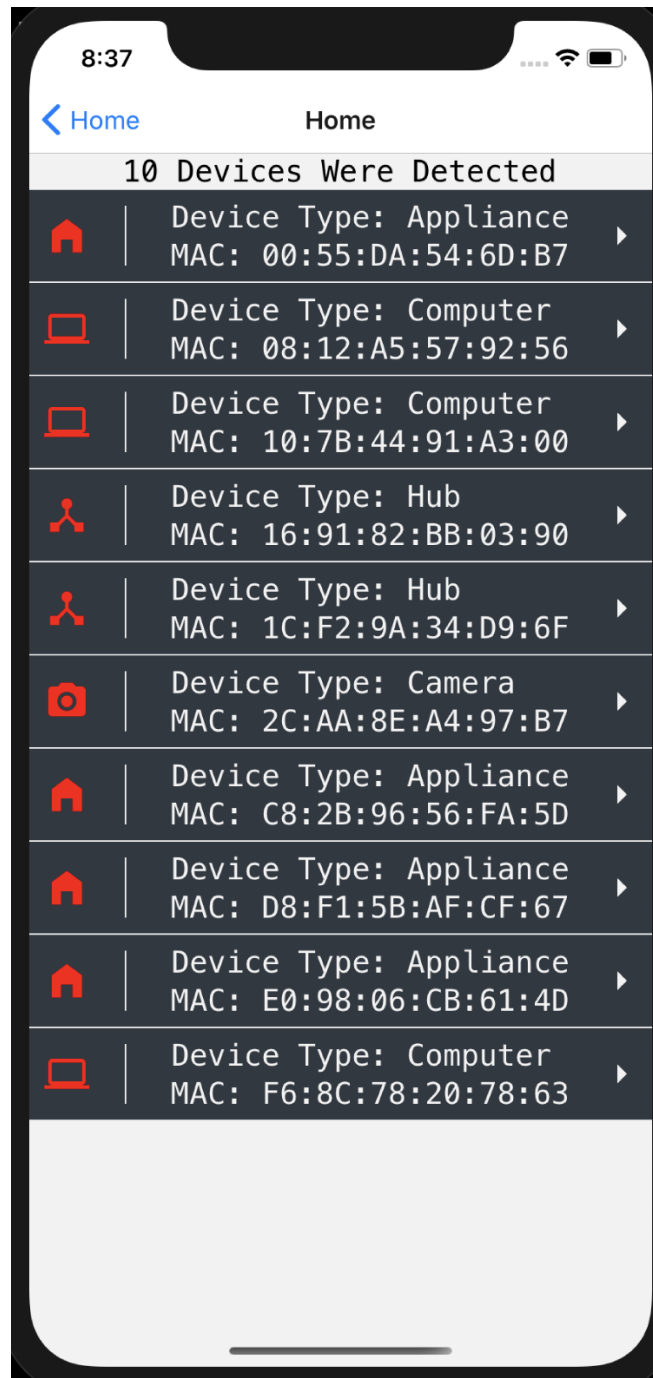


Figure 42: Second page in the beginning

The third page is the device info page. It shows extra details about the device that was selected from the previous page. The information that we deemed to be of importance for the user are; device type, device name, MAC address, device active time, protocols used by the device, servers that the device communicates with, the average size of the sent packets, data flow rate, and data flow volume. The banner on the top of the page is either green or red depending on whether the device is safe or not. We implemented two buttons in the bottom of the page for the user to choose whether they recognize the device or not. If the user recognizes the device and presses on the known device button, the banner's color will change to green, the database will be updated accordingly and said device will be marked safe. Otherwise, if the user does not recognize the device, and presses on the unknown device button, the banner's color will change to red and the database will be updated accordingly.

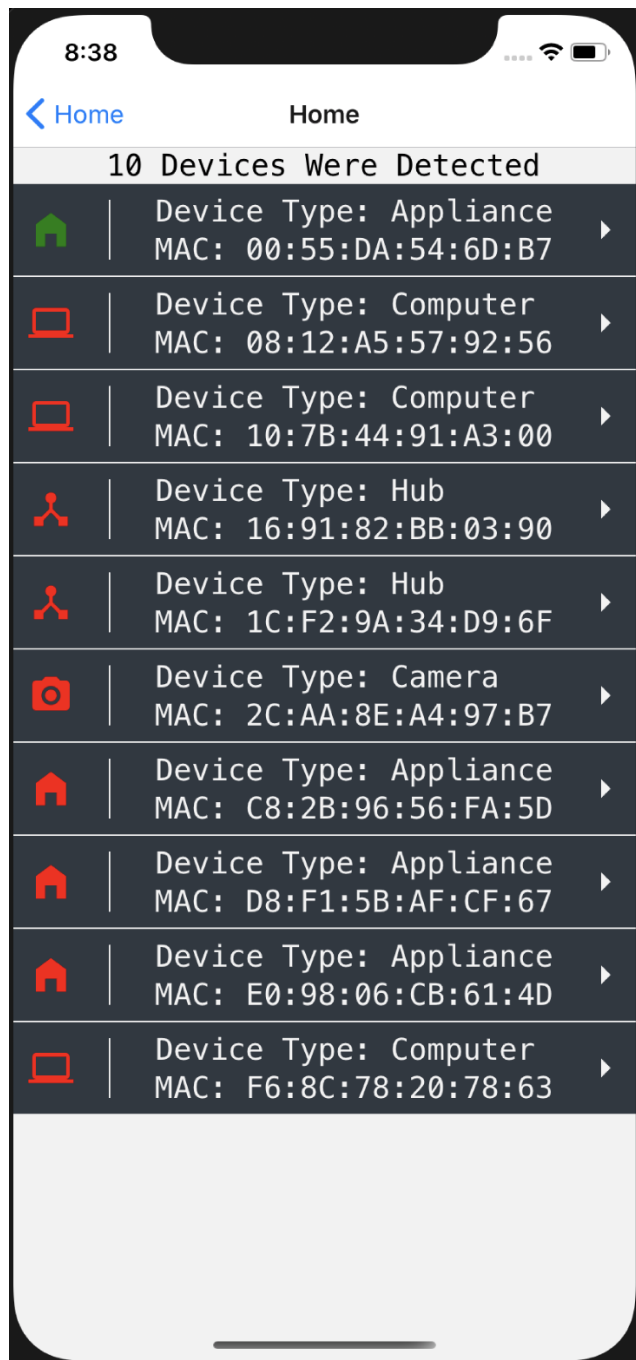


Figure 43: Second page after marking a safe device

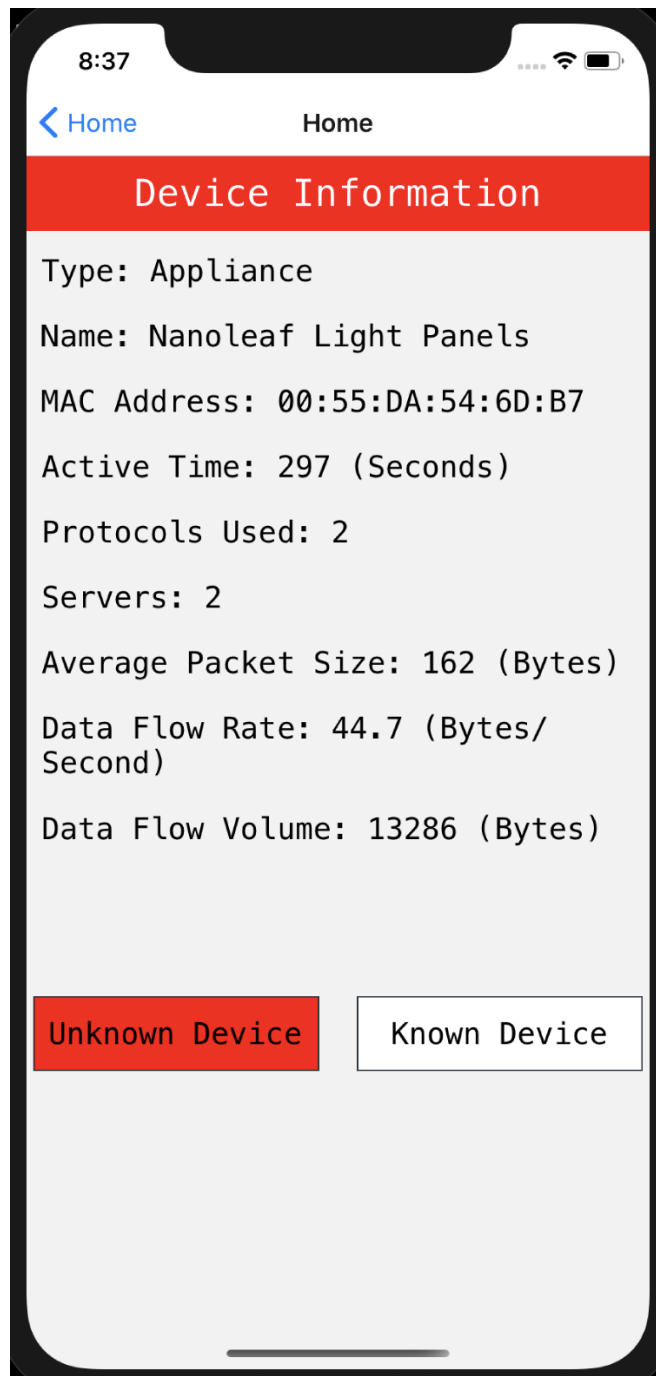


Figure 44: Device info when it is unsafe

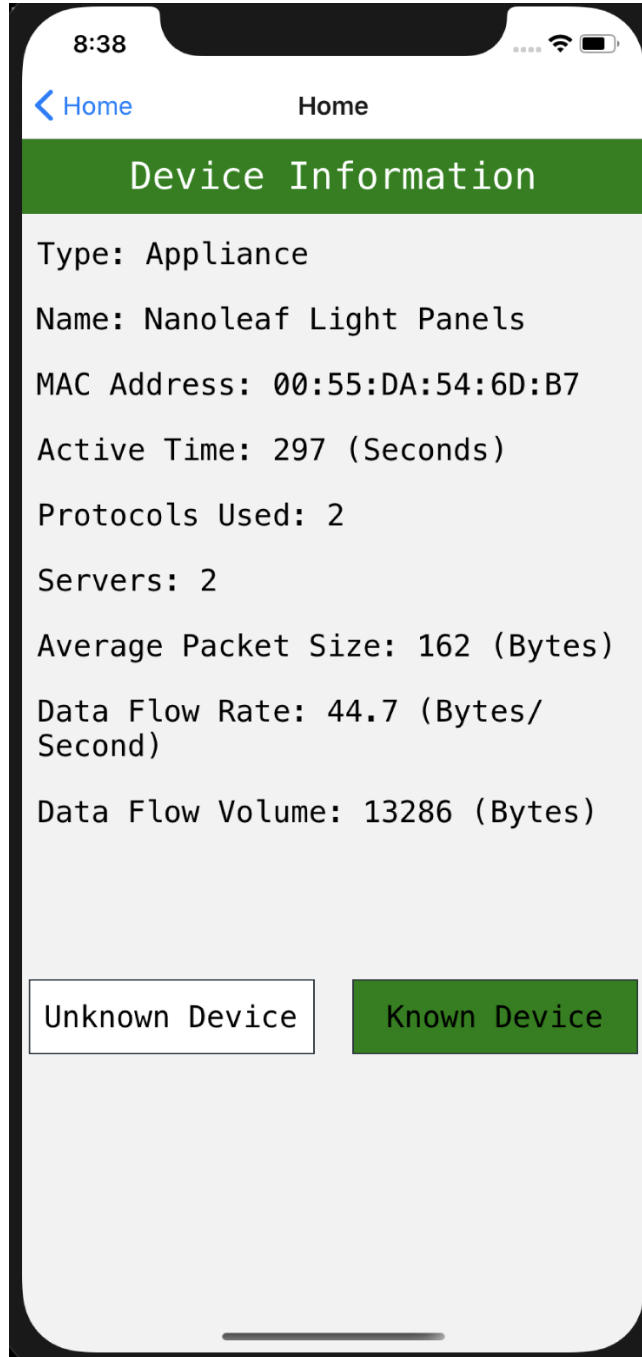


Figure 45: Device info when it is safe

9.7.1 Limitations and Future Work

For future enhancement, the app could have the functionality to send a push notification to the user once a new device is connected, so the user can be notified that there is a possible intruder in their area and open the app and check the device and mark it as safe or unsafe. Another function that could be added is an option for the user, if a malicious device was detected, is to disconnect the malicious device from the network.

10 Conclusion

In conclusion, our project solves a problem that no other literature tackled which is not only detecting the devices but also allowing the user to know if there is a new rogue device in their area. Our project consists of three main stages; data acquisition, machine learning prediction and showing the output to the user and alerting them. To bridge between the three stages we used the back-end service Firebase, where we created a real-time database on the cloud that allowed the different components of the project to communicate. We used Python and React Native as the main development tools for our project, and we used ASUS's Tinker Board as the hardware component of our project.

We truly believe that the project has great potential, especially in the industry as it solves a real life problem. We also believe that there is a big room for improvement, especially for the machine learning model and its accuracy, as well as the interface of the application and the whole output system, as they all can be optimised and become more efficient. To achieve that we need to put more efforts into communicating with the industry and training the machine learning model with bigger data sets that contain a wider variety of devices.

References

- [1] M. R. Shahid, G. Blanc, Z. Zhang, and H. Debar, "IoT Devices Recognition Through Network Traffic Analysis," 2018 IEEE International Conference on Big Data (Big Data), 2018.
- [2] "State of the IoT 2018: Number of IoT devices now at 7B - Market accelerating," IoT Analytics, 08-Aug-2018. [Online]. Available: <https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/>. [Accessed: 15-Mar-2020].
- [3] L. Bai, L. Yao, S. S. Kanhere, X. Wang, and Z. Yang, "Automatic Device Classification from Network Traffic Streams of Internet of Things," 2018 IEEE 43rd Conference on Local Computer Networks (LCN), 2018.
- [4] C. Fachkha, E. Bou-Harb, A. Boukhtouta, S. Dinh, F. Iqbal and M. Debbabi, "Investigating the dark cyberspace: Profiling, threat-based analysis and correlation," 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS), Cork, 2012, pp. 1-8.
- [5] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics," IEEE Transactions on Mobile Computing, vol. 18, no. 8, pp. 1745–1759, Jan. 2019.
- [6] A. Hamza, H. H. Gharakheili, T. A. Benson, and V. Sivaraman, "Detecting Volumetric Attacks on IoT Devices via SDN-Based Monitoring of MUD Activity," Proceedings of the 2019 ACM Symposium on SDN Research, Mar. 2019.
- [7] Cisco, 2020. [Online]. Available: <https://github.com/cisco/joy>

- [8] Let's get clear about react native, 2018. [Online]. Available: <https://www.simplytechnologies.net/blog/2018/4/4/lets-get-clear-about-react-native>
- [9] "Object Storage Features – Amazon S3", Amazon Web Services, Inc., 2019. [Online]. Available: <https://aws.amazon.com/s3/features/>. [Accessed: 05-May- 2020].
- [10] "Detect and Respond to IoT events", Amazon Web Services, Inc., 2019. [Online]. Available: <https://aws.amazon.com/iot-events/>. [Accessed: 05- May- 2020].
- [11] "AWS IoT Core Features - Amazon Web Services", Amazon Web Services, Inc., 2019.[Online].Available: <https://aws.amazon.com/iot-core/features/?nc=sn&loc=3>. [Accessed: 05- May- 2020].
- [12] Data Collected for ACM SOSR 2019. (n.d.). Retrieved November 26, 2020, from <https://iotanalytics.unsw.edu.au/attack-data>
- [13] Pandas Basics - Learn Python - Free Interactive Python Tutorial. (n.d.). Retrieved November 26, 2020, from https://www.learnpython.org/en/Pandas_Basics
- [14] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, É. (1970, January 01). Scikit-learn: Machine Learning in Python. Retrieved November 26, 2020, from <https://jmlr.org/papers/v12/pedregosa11a.html>

Appendix A Data Acquisition and Preprocessing

A.1 Shell Script

```
PATH=$PATH:/sbin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/home/chaos/joy/bin
```

```
ethr_addr_list="08:12:A5:57:92:56  
               00:55:DA:54:6D:B7  
               C8:2B:96:56:FA:5D  
               6C:E8:C6:38:B3:F1  
               D8:F1:5B:AF:CF:67  
               1C:F2:9A:34:D9:6F  
               2C:AA:8E:A4:97:B7  
               F6:8C:78:20:78:63  
               E0:98:06:CB:61:4D  
               00:17:88:B3:4A:D0  
               3C:52:82:2D:1C:5B  
               10:7B:44:91:A3:00  
               16:91:82:BB:03:90"
```

```
fileCo=1  
while [ $fileCo -le 1 ]  
do  
    fileName="${fileCo}.pcap"  
    echo Processing $fileName  
    deviceCo=1
```

```

for addr in $ethr_addr_list
do
    joy output="json_files/${fileCo}_${deviceCo}.json"\
        bpf="ether host ${addr}"\
        bidir=1\
        dns=1\
        http=1\
        "pcap_files/${fileCo}.pcap"
    ((deviceCo++))
done
((fileCo++))
done

echo Running HARPY on 5-minute interval
python3 harpy.py -t 5
echo Running HARPY on 15-minute interval
python3 harpy.py -t 15
echo Running HARPY on 30-minute interval
python3 harpy.py -t 30

echo Done

```

A.2 Harpy Shell Launcher

```

#
#

```



```

#           PROJECT HARPY - HBKU Final Year Project
#
#   Data Collection, Preprocessing and Launcher Developer: Selman Tabet
#   Machine Learning Developer: Mohamed Amara
#   Frontend and App Developer: Omar Elshal
#   Supervisor: Dr. Ala Al-Fuqaha
#
#
# ---- Prerequisite file structure ----
# - Harpy directory           // Main app directory (MANDATORY, most essential)
# |_ /pcap_files/            // Capture files (MANDATORY - for Joy Tool Shell)
# |_ /csv_files/             // (Default) Output of the feature extraction
# |   |_ ListCSV.csv         // List of MAC addresses (Optional iff the abs)
# |_ /json_files/           // Where Joy JSON output is stored (MANDATORY)
#   |_ whois_record.json    // For RDAP requests in extract_features.py (M)
#
# DO NOT RUN UNLESS THE MANDATORY FILES AND FOLDERS IN THE FILE TREE ARE COMPLETE
#
#
#
# Changelog: HARPY V3.4.2 - Style Update.
#
# [STYLE]
# - Moved changelog to harpy.py

```

```

# - Additional harpy.py Launcher styling.
#
#
# [ROADMAP]
#
# * Next Beta branch, in anticipation of new datasets:
#   ~ Have harpy.py output the number of MACs in ListCSV.csv, this would be handed
#   to the device_co for loop to avoid hardcoding values.
#   ~ For the same reason previously stated, the number of capture files needs to
#   be dynamically adjusted for the files in /pcap_files/ directory.
#   ~ New Python script may be developed to call cisco/joy for all MACs in ListCSV.csv
#   due to the potential usage of os.system(joy <args>), unless the user has "joy"
#   ~ In the event that the WHOIS Record was found to be outdated, it shall be checked
#
#

import argparse
import os
import extract_features as extract

dir_path = os.path.dirname(os.path.realpath(__file__))

parser = argparse.ArgumentParser(description='Project HARPY Preprocessor: Takes Joy
parser.add_argument('-t', action='store', type = int, default = 5, dest='parsed_inte
parser.add_argument('-p', action='store', type = str, default = dir_path, dest='pars

```

```

parser.add_argument('-l', action='store', type = str, default = os.path.join(dir_path, 'list.csv'))
parser.add_argument('-o', action='store', type = str, default = os.path.join(dir_path, 'output.csv'))
args = parser.parse_args()

print(" ")
print("HARPY Path: " + args.parsed_path)
print(" ")
print("ListCSV.csv Directory: " + args.parsed_MAClist)
print(" ")
print("Output Path: " + args.parsed_output)
print(" ")

extract.mac_map(args.parsed_MAClist, extract.extract_features(args.parsed_path, args.parsed_output))

```

A.3 Feature Extraction Script

```

# HARPY Joy JSON Feature Extraction
#
# Created by: Selman Tabet (@selmantabet - https://selman.io/)
# Special thanks to Dr. Ihab Mohammed from Western Michigan University for showing

import os
import json
import datetime
import csv
import sys

```

```
import pandas as pd
import json
from ipwhois import IPWhois
'''
```

Features List:

Total Sleep Time: is the total time of no activity

Total Active Time: is the total time of activity

Total Flow Volume: number of bytes (sent/received) by the IoT device

Flow Rate: Total Flow Volume / Total Active Time

Average Packet Size (number of bytes sent/received / number of packets sent/recieved)

Number of servers (excluding DNS (53) and NTP (123))

RDAP/WHOIS Registration Name along with number of flows for each resolved name. Stored in a dictionary

Number of protocols (based on destination port number)

Number of unique DNS requests. (based on qn and rn in joy tool)

DNS Interval: total time for using DNS

```
NTP interval: total time for using NTP
```

```
'''
```

```
def extract_features(path, interval_time): # Installation path and and integer interval
    print("Selected time interval: " + str(interval_time) + " minutes")
    period_time = 60 * interval_time # Convert from minutes to seconds
    app_directory = path
    features_file_name = os.path.join(app_directory, 'csv_files', 'output_' + str(interval_time) + '.csv')

    # Prepare writer to write extracted features for device_co
    features_csv = open(features_file_name, 'w')
    features_writer = csv.writer(features_csv, delimiter=',')

    features_writer.writerow(['total_sleep_time', 'total_active_time', 'total_flow_volume'])
    overall_ports_dict = {}
    overall_dns_dict = {}
    with open(os.path.join(app_directory, 'json_files', 'whois_record.json')) as f:
        whois_record = json.load(f)

    for device_co in range(1, 32):
        # Process all days for device_co
        for day_co in range(1, 21):
            # Read flows from flows file containing 1 day data
            flows_file_name = os.path.join(app_directory, 'json_files', str(day_co) + '.json')
```

```
print('Processing ' + flows_file_name + ' ...')
flows_file = open(flows_file_name, 'r')
flows = flows_file.readlines()
flows_file.close()

# Features list
total_sleep_time = 0
total_active_time = 0

total_flow_volume = 0
total_packets = 0

servers_dic = {}
rdap_asn_record = {"Not Resolved":0}
ports_dic = {}
dns_query_dic = {}

dns_interval = 0
ntp_interval = 0

# Time management variables
prev_end_time = 0
period_start_time = 0
period_flow_co = 0
```

```

# Skip the first record, which stores Joy configurations used
nu_of_flows = len(flows)
for flow_co in range(1, nu_of_flows):
    # Collect features
    #print(flow_co)
    flow_data = json.loads(flows[flow_co])
    # Get times
    cur_start_time = datetime.datetime.utcnow().timestamp()
    cur_end_time = datetime.datetime.utcnow().timestamp()
    cur_total_seconds = int( (cur_end_time - cur_start_time).total_seconds())
    total_active_time += cur_total_seconds

    # Get number of bytes and number of packets going out
    total_flow_volume += int(flow_data['bytes_out'])
    total_packets += int(flow_data['num_pkts_out'])
    # Get number of bytes and number of packets going in
    if 'bytes_in' in flow_data:
        total_flow_volume += int(flow_data['bytes_in'])
        total_packets += int(flow_data['num_pkts_in'])

    # Get source port
    #if flow_data['sp'] is not None:
    #     port = int(flow_data['sp'])
    #     if port not in ports_dic:
    #         ports_dic[port] = 1

```

```

#     else:
#         ports_dic[port] += 1

# Get destination port
port = 0
if flow_data['dp'] is not None:
    port = int(flow_data['dp'])
    if port not in ports_dic:
        ports_dic[port] = 1
    else:
        ports_dic[port] += 1
'''
# Overall ports dictionary
if port not in overall_ports_dict:
    overall_ports_dict[port] = 1
else:
    overall_ports_dict[port] += 1
'''

if port == 53:
    dns_interval += cur_total_seconds
elif port == 123:
    ntp_interval += cur_total_seconds

# Get the server and WHOIS Record
if port != 53 and port != 123:

```



```

server = flow_data['da']

#print("Inspecting IP: " + server)

if server not in whois_record:

    try:

        ip_query = IPWhois(server)

        RDAP = ip_query.lookup_rdap(depth=1, rate_limit_timeout=

        server_id = RDAP["asn_description"] # From what I've se

        whois_record[server] = server_id # Store info into WHOI

        #print("New IP resolved as " + server_id)

    except:

        #print("Not resolved...")

        server_id = "Not Resolved"

        whois_record[server] = server_id # Though technically i

else:

    server_id = whois_record[server] # The IP was previously lo

if server not in servers_dic:

    rdap_asn_record[server_id] = 1

    servers_dic[server] = 1

else:

    servers_dic[server] += 1

    rdap_asn_record[server_id] += 1

# Get DNS query

if 'dns' in flow_data:

```

```

for dns_query in flow_data['dns']:
    #print(dns_query)
    query = ''
    if 'qn' in dns_query:
        query = dns_query['qn']
    elif 'rn' in dns_query:
        query = dns_query['rn']

    if query != '':
        if query not in dns_query_dic:
            dns_query_dic[query] = 1
        else:
            dns_query_dic[query] += 1
        ...

        # Overall dns query dictionary
        if query not in overall_dns_dict:
            overall_dns_dict[query] = 1
        else:
            overall_dns_dict[query] += 1
        ...

if flow_co == 1:
    period_start_time = cur_start_time

if period_flow_co == 0:

```

```

    period_flow_co = 1
else:
    period_flow_co += 1
    cur_sleep_time = int( (cur_start_time - prev_end_time).total_seconds() )
    # Could be 0 for overlapping flows (device is active)
    if cur_sleep_time > 0:
        total_sleep_time += cur_sleep_time

if (int( (cur_end_time - period_start_time).total_seconds() ) >= 0):
    # Finalize features computations
    flow_rate = 0
    if total_active_time > 0:
        flow_rate = total_flow_volume / total_active_time
    avg_packet_size = 0
    if total_packets > 0:
        avg_packet_size = total_flow_volume / total_packets

    # Save features
    features_writer.writerow([total_sleep_time, total_active_time, flow_rate, avg_packet_size])

    # Reinitialize features
    total_sleep_time = 0
    total_active_time = 0
    total_flow_volume = 0
    total_packets = 0

```

```

        servers_dic = {}
        rdap_asn_record = {"Not Resolved":0}
        ports_dic = {}
        dns_query_dic = {}
        dns_interval = 0
        ntp_interval = 0
        period_flow_co = 0
        period_start_time = cur_end_time

    prev_end_time = cur_end_time

features_csv.close()
with open(os.path.join(app_directory, 'json_files', 'whois_record.json'), 'w') as f:
    json.dump(whois_record, f)
return features_file_name

```

```

def mac_map(mac_source, mac_target, output):
    print("Mapping MACs...")
    mac_src = pd.read_csv(mac_source)
    mapping = pd.read_csv(mac_target)
    mapped = mapping.merge(mac_src, on='device_co', how='left').rename(columns={'MAC'})
    mapped.to_csv(os.path.join(output, os.path.basename(mac_target).split(".csv")[0]

```

```

    print("Done.")
    return 0

'''

stats_csv = open(stats_file_name, 'w')
stats_writer = csv.writer(stats_csv, delimiter=',')
stats_writer.writerow([overall_ports_dict, overall_dns_dict])
stats_csv.close()
'''

```

Appendix B Machine Learning Code

```

from sklearn.ensemble import RandomForestClassifier
from sklearn import model_selection
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn import preprocessing
from mac_vendor_lookup import MacLookup
from sklearn.preprocessing import OneHotEncoder # use OneHotEncoder
from sklearn.compose import ColumnTransformer
import pyrebase
import random
import time

```

```

# 1) Prepare the data for training
# 2) Train the model
# 3) Prepare the testing data
# 4) Extract the time scale from the database
# 5) Choose the module to use and train it with the respective dataset
# 6) Go over the captured data in the sheet and
#predict the devices with
the ML model
# 7) Update the firebase db with the predicted devices

# Data preparation:

# load csv data

# Training set for the 5 min time scale
df_5m_unsw = pd.read_csv("unsw_dataset_features_mapped_5m.csv",
low_memory=False)
# Training set for the 15 min time scale
df_15m_unsw = pd.read_csv("unsw_dataset_features_mapped_15m.csv",
low_memory=False)
# Training set for the 30 min time scale
df_30m_unsw = pd.read_csv("unsw_dataset_features_mapped_30m.csv",
low_memory=False)

df_5m = pd.read_csv("output_5m_mapped.csv", low_memory=False)

```

```

# Training set for the 5 min time scale
df_15m = pd.read_csv("output_15m_mapped.csv", low_memory=False)
# Training set for the 15 min time scale
df_30m = pd.read_csv("output_30m_mapped.csv", low_memory=False)
# Training set for the 30 min time scale

# counts the number of devices detected
print(df_5m_unsw["Device Type"].value_counts(sort=1))
numOfDevices = len(df_5m_unsw["Device Type"].value_counts(sort=1))
print("Number of types detected is: " + str(numOfDevices) + " types")
print("\n")

# remove the unnecessary data columns
df_5m.drop(["device_co", "Device Name", "MAC_address"], axis=1,
inplace=True)
df_15m.drop(["device_co", "Device Name", "MAC_address"], axis=1,
inplace=True)
df_30m.drop(["device_co", "Device Name", "MAC_address"], axis=1,
inplace=True)

# handle missing values
df_5m = df_5m.dropna()
df_15m = df_15m.dropna()
df_30m = df_30m.dropna()
#pred = pred.dropna()

```

```

# define the dependent variable
Y5m = df_5m["Device Type"].values
Y15m = df_15m["Device Type"].values
Y30m = df_30m["Device Type"].values

# convert strings into numbers so the model can process it
le = preprocessing.LabelEncoder() # use LabelEncoder
df_5m['rdap_asn'] = le.fit_transform(df_5m['rdap_asn'])
df_5m['Connection Type'] = le.fit_transform(df_5m['Connection Type'])
df_15m['rdap_asn'] = le.fit_transform(df_15m['rdap_asn'])
df_15m['Connection Type'] = le.fit_transform(df_15m['Connection Type'])
df_30m['rdap_asn'] = le.fit_transform(df_30m['rdap_asn'])
df_30m['Connection Type'] = le.fit_transform(df_30m['Connection Type'])

# define the independent variable
X5m = df_5m.drop(labels=["Device Type"], axis=1)
X15m = df_15m.drop(labels=["Device Type"], axis=1)
X30m = df_30m.drop(labels=["Device Type"], axis=1)

# split data into training set and testing set
from sklearn.model_selection import train_test_split

# 5 minutes time scale model

```



```

X_train_5m, X_test_5m, Y_train_5m, Y_test_5m = train_test_split(X5m, Y5m,
test_size=0.2, random_state=10, shuffle=True)

# 15 minutes time scale model
X_train_15m, X_test_15m, Y_train_15m, Y_test_15m = train_test_split(X15m, Y15m,
test_size=0.2, random_state=10, shuffle=True)

# 30 minutes time scale model
X_train_30m, X_test_30m, Y_train_30m, Y_test_30m = train_test_split(X30m, Y30m,
test_size=0.2, random_state=10, shuffle=True)

# create the model
from sklearn.ensemble import RandomForestClassifier

model_5m = RandomForestClassifier(n_estimators = 10, random_state=30)
model_15m = RandomForestClassifier(n_estimators = 10, random_state=30)
model_30m = RandomForestClassifier(n_estimators = 10, random_state=30)

# train the model
model_5m.fit(X_train_5m, Y_train_5m)
model_15m.fit(X_train_15m, Y_train_15m)
model_30m.fit(X_train_30m, Y_train_30m)

# test the model
prediction_test_5m = model_5m.predict(X_test_5m)
prediction_test_15m = model_15m.predict(X_test_15m)

```

```
prediction_test_30m = model_30m.predict(X_test_30m)
```

```
from sklearn import metrics
```

```
print("Testing results for the 5 minutes time scale model...")
```

```
print("Accuracy = ",
```

```
metrics.accuracy_score(Y_test_5m, prediction_test_5m))
```

```
# check the contribution of each independent variable
```

```
feature_list = list(X5m.columns)
```

```
feature_imp = pd.Series(model_5m.feature_importances_,  
index=feature_list).sort_values(ascending=False)
```

```
print(feature_imp)
```

```
print("\n\n\n")
```

```
print("Testing results for the 15 minutes time scale model...")
```

```
print("Accuracy = ", metrics.accuracy_score(Y_test_15m,  
prediction_test_15m))
```

```
# check the contribution of each independent variable
```

```
feature_list = list(X15m.columns)
```

```
feature_imp = pd.Series(model_15m.feature_importances_,  
index=feature_list).sort_values(ascending=False)
```

```
print(feature_imp)
```

```

print("\n\n\n")

print("Testing results for the 30 minutes time scale model...")
print("Accuracy = ", metrics.accuracy_score(Y_test_30m,
prediction_test_30m))

# check the contribution of each independent variable
feature_list = list(X30m.columns)
feature_imp = pd.Series(model_30m.feature_importances_,
index=feature_list).sort_values(ascending=False)
print(feature_imp)

# firebase configuration

config = {
    "apiKey": "AIzaSyBJqqj1jgw0sNgCD-iC_uKbKPLqdg5Ytp8",
    "authDomain": "harpy-c8519.firebaseio.com",
    "databaseURL": "https://harpy-c8519.firebaseio.com/",
    "projectId": "harpy-c8519",
    "storageBucket": "harpy-c8519.appspot.com"
}

firebase = pyrebase.initialize_app(config)
db = firebase.database()
ID = '-MHfQDfcv017vmKEFU0c'

```

```

timeScale = db.child(ID).get().val()['timeScale']
print(timeScale)
print("\n")

chosenModel = ''
captured_devices = {}

# A sample of a captured dataset
captured_set_sample = pd.read_csv("test_output.csv")
captured_set_sample = captured_set_sample.dropna()
captured_set_sample_enc = captured_set_sample[0:]

le = preprocessing.LabelEncoder() # use LabelEncoder
captured_set_sample_enc['rdap_asn'] = le.fit_transform(
captured_set_sample_enc['rdap_asn'])
captured_set_sample_enc['Connection Type'] = le.fit_transform(
captured_set_sample_enc['Connection Type'])
captured_set_sample_enc.drop(
["device_co", "Device Name", "MAC_address", "Device Type"], axis=1,
inplace=True)

while (1):

```

```

db_content = []
timeScale = db.child(ID).get().val()['timeScale']
if (timeScale == 5):
    chosenModel = model_5m
elif (timeScale == 15):
    chosenModel = model_15m
elif (timeScale == 30):
    chosenModel = model_30m

for row1, row2 in zip(captured_set_sample_enc.iterrows(),
captured_set_sample.iterrows()):
    features = list(row2)[1]
    if (not first_row):
        if (features[13].strip() not in db_content):
            device_type = chosenModel.predict(list(row1)[1:12])

            db_content+= [features[13].strip()]
            if(device_type[0]=='Hub'):
                iconName = 'device-hub'

            elif(device_type[0]=='Camera'):
                iconName = 'camera-alt'

            elif(device_type[0]=='Computer'):
                iconName = 'computer'

```

```
elif(device_type[0]=='Appliance'):
    iconName = 'home-filled'

elif(device_type[0]=='Sensor'):
    iconName = 'settings-remote'

elif(device_type[0]=='Misc'):
    iconName = 'miscellaneous-services'

elif(device_type[0]=='Network'):
    iconName = 'router-wireless'

db.child(ID).child(features[13].strip()).update({
    "Device Type": device_type[0],
    "Device MAC": features[13].strip(),
    "Device Name": features[12].strip(),
    "Device Total Active Time": features[1],
    "Device Total Flow Volume": features[2],
    "Device Flow Rate": features[3],
    "Device Avarage Packet Size": features[4],
    "Number of Servers": features[5],
    "Number of Protocols Used": features[6],
    "iconName": iconName,
    "safety": False,
```

```
})
```

Appendix C React Native Code

C.1 Navigation

```
import {createAppContainer} from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack';
import HomeScreen from './src/screens/HomeScreen';
import ListScreen from './src/screens/ListScreen';
import DeviceInfo from './src/screens/DeviceInfo';

const navigator = createStackNavigator(
  {
    Home: HomeScreen,
    List: ListScreen,
    Device: DeviceInfo,
  },
  {
    initialRouteName: 'Home',
    defaultNavigationOptions: {
      title: 'Home',
    },
  },
);
```

```
export default createAppContainer(navigator);
```

C.2 Firebase Configuration

```
import * as firebase from 'firebase';

let config = {
  apiKey: 'AIzaSyBJqqj1jgw0sNgCD-iC_uKbKPLqdg5Ytp8',
  authDomain: 'harpy-c8519.firebaseio.com',
  databaseURL: 'https://harpy-c8519.firebaseio.com/',
  projectId: 'harpy-c8519',
  storageBucket: 'harpy-c8519.appspot.com',
};

firebase.initializeApp(config);

export default firebase;
```

C.3 Home Screen

```
import React from 'react';
import {Text, StyleSheet, View, TouchableOpacity, Image} from 'react-native';
import firebase from '../config';
import {createAppContainer} from 'react-navigation';
import {createStackNavigator} from 'react-navigation-stack';
import ListScreen from './ListScreen';

class HomeScreen extends React.Component {
```



```

state = {
  buttonBGColorsArr: ['#303841', '#f3f3f3', '#f3f3f3'],
  buttonFontColorsArr: ['#f3f3f3', '#303841', '#303841'],
};

render() {
  return (
    <View style={styles.viewStyle}>
      <View style={{flex: 1, marginTop: 10}}>
        <Image
          style={{
            justifyContent: 'center',
            width: 400,
            height: 400,
            resizeMode: 'contain',
            alignSelf: 'center',
          }}
          source={require('../../assets/logo.png')}
        />
      </View>
      <View style={{flex: 1, marginTop: 50}}>
        <Text style={styles.text}>
          Select your scan time scale in minutes
        </Text>
        <View style={{flex: 1, flexDirection: 'row', marginTop: -150}}>
          <TouchableOpacity

```

```

style={{
  padding: 10,
  backgroundColor: this.state.buttonBGColorsArr[0],
  marginTop: 50,
  width: 80,
  alignSelf: 'center',
  margin: 20,
  borderWidth: 1,
  borderColor: '#303841',
}}
onPress={() => {
  this.setState({
    buttonBGColorsArr: ['#303841', '#f3f3f3', '#f3f3f3'],
    buttonFontColorsArr: ['#f3f3f3', '#303841', '#303841'],
  });
  firebase
    .database()
    .ref('/-MHfQDfcv017vmKEFU0c/timeScale')
    .set(5);
}}>
<Text
  style={{
    fontSize: 20,
    color: this.state.buttonFontColorsArr[0],
    textAlign: 'center',

```

```

    }}>

    5
  </Text>
</TouchableOpacity>
<TouchableOpacity
  style={{
    padding: 10,
    backgroundColor: this.state.buttonBGColorsArr[1],
    marginTop: 50,
    width: 80,
    alignSelf: 'center',
    margin: 20,
    borderWidth: 1,
    borderColor: '#303841',
  }}
  onPress={() => {
    this.setState({
      buttonBGColorsArr: ['#f3f3f3', '#303841', '#f3f3f3'],
      buttonFontColorsArr: ['#303841', '#f3f3f3', '#303841'],
    });
    firebase
      .database()
      .ref('/-MHfQDfcv017vmKEFU0c/timeScale')
      .set(15);
  }}>

```

```

<Text
  style={{
    fontSize: 20,
    color: this.state.buttonFontColorsArr[1],
    textAlign: 'center',
  }}>
  15
</Text>
</TouchableOpacity>
<TouchableOpacity
  style={{
    padding: 10,
    backgroundColor: this.state.buttonBGColorsArr[2],
    marginTop: 50,
    width: 80,
    alignSelf: 'center',
    margin: 20,
    borderWidth: 1,
    borderColor: '#303841',
  }}
  onPress={() => {
    this.setState({
      buttonBGColorsArr: ['#f3f3f3', '#f3f3f3', '#303841'],
      buttonFontColorsArr: ['#303841', '#303841', '#f3f3f3'],
    });
  }}

```

```

        firebase
            .database()
            .ref('/-MHfQDfcv017vmKEFU0c/timeScale')
            .set(30);
    }}>
    <Text
        style={{
            fontSize: 20,
            color: this.state.buttonFontColorsArr[2],
            textAlign: 'center',
        }}>
        30
    </Text>
</TouchableOpacity>
</View>
</View>
<View style={{flex: 1, marginTop: -100}}>
    <Text style={styles.text}>
        Press scan to show the devices around you
    </Text>
    <TouchableOpacity
        style={styles.buttonStyle}
        onPress={() => this.props.navigation.navigate('List')}>
        <Text style={styles.textStyle}>Scan</Text>
    </TouchableOpacity>

```

```
        </View>
    </View>
);
}
}
```

```
const styles = StyleSheet.create({
  text: {
    fontSize: 20,
    textAlign: 'center',
    fontFamily: 'Menlo',
  },
  viewStyle: {
    flex: 1,
    alignItems: 'center',
  },
  textStyle: {
    fontSize: 20,
    color: '#ffffff',
    textAlign: 'center',
    fontFamily: 'Menlo',
  },
  buttonStyle: {
    padding: 10,
    backgroundColor: '#303841',
  },
});
```

```
        width: 200,
        alignSelf: 'center',
        marginTop: 35,
    },
});

export default HomeScreen;
```

C.4 List Screen

```
import {
    View,
    Text,
    StyleSheet,
    FlatList,
    TouchableOpacity,
    SafeAreaView,
} from 'react-native';
import firebase from '../config';
import Icon from 'react-native-vector-icons/MaterialIcons';
var data = [{1: 1}];

class ListScreen extends React.Component {
    state = {
        listData: [],
    };
};
```

```

componentDidMount() {
  firebase
    .database()
    .ref('/-MHfQDfcv017vmKEFU0c')
    .on('value', (querySnapshot) => {
      data = [querySnapshot.val()];
      this.setState({listData: data});
    });
}

render() {
  return (
    <SafeAreaView>
      <View>
        <Text
          style={{
            fontFamily: 'Menlo',
            alignSelf: 'center',
            color: 'black',
            fontSize: 20,
          }}>
          {Object.keys(data[0]).length - 1} Devices Were Detected
        </Text>
        <FlatList
          data={this.state.listData}

```



```

renderItem={({item}) => {
  return (
    <SafeAreaView style={styles.container}>
      {Object.keys(item).map((value, index) => {
        return (
          value !== 'timeScale' &&
          value !== 'scan' && (
            <TouchableOpacity
              onPress={() =>
                this.props.navigation.navigate('Device', {
                  device: item[value],
                })
              }
              style={styles.buttonStyle}>
              <Icon
                name={item[value].iconName}
                size={30}
                color={item[value].safety ? 'green' : 'red'}
                style={styles.iconStyle}
              />
              <View style={styles.iconSeparator} />
              <View>
                <Text style={styles.textStyle}>
                  Device Type: {item[value]['Device Type']}
                </Text>

```

```

        <Text style={styles.textStyle}>MAC: {value}</Text>
    </View>

    <Icon
      name={'arrow-right'}
      size={30}
      color="#f3f3f3"
      style={{
        alignSelf: 'center',
      }}
    />
  </TouchableOpacity>
)
);
}}
</SafeAreaView>
);
}}
/>
</View>
</SafeAreaView>
);
}
}
}

```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#f3f3f3',
  },
  textStyle: {
    fontSize: 20,
    color: '#f3f3f3',
    padding: 1,
    marginLeft: 10,
    fontFamily: 'Menlo',
  },
  buttonStyle: {
    flexDirection: 'row',
    alignItems: 'center',
    justifyContent: 'space-between',
    padding: 5,
    backgroundColor: '#303841',
    marginTop: 1,
    width: '100%',
    flex: 1,
  },
  iconSeparator: {
```

```
    backgroundColor: '#f3f3f3',
    width: 1,
    height: 40,
  },
  iconStyle: {
    padding: 5,
  },
});
```

```
export default ListScreen;
```

C.5 Device info page

```
import React from 'react';
import {View, Text, StyleSheet, TouchableOpacity} from 'react-native';
import firebase from '../config';
var data;
var color;

class DeviceInfo extends React.Component {
  state = {
    known: this.props.navigation.getParam('device')['safety'],
  };

  componentDidMount() {
    const item = this.props.navigation.getParam('device');
```

```

firebase
  .database()
  .ref(`/-MHfQDfcv017vmKEFU0c/${item['Device MAC']}/safety`)
  .on('value', (querySnapshot) => {
    data = [querySnapshot.val()];
    this.setState({known: data});
    var color = item['safety'];
  });
}

render() {
  const item = this.props.navigation.getParam('device');
  var headerbgColor = color ? 'green' : 'red';
  var unknownbgColor = color ? 'white' : 'red';
  var knownbgColor = color ? 'green' : 'white';

  const toggleSwitch = () => {
    firebase
      .database()
      .ref(`/-MHfQDfcv017vmKEFU0c/${item['Device MAC']}/safety`)
      .set(!item['safety']);
    this.setState({known: !this.state.known});
    color = !color;
  };
}

```

```

return (
  <View style={styles.viewStyle}>
    <Text
      style={{
        fontSize: 25,
        color: 'white',
        backgroundColor: headerbgColor,
        textAlign: 'center',
        padding: 10,
        borderRadius: 5,
        fontFamily: 'Menlo',
        marginBottom: 5,
      }}>
      Device Information
    </Text>
    <Text style={styles.textStyle}>Type: {item['Device Type']}</Text>
    <Text style={styles.textStyle}>Name: {item['Device Name']}</Text>
    <Text style={styles.textStyle}>MAC Address: {item['Device MAC']}</Text>
    <Text style={styles.textStyle}>
      Active Time: {item['Device Total Active Time']} (Seconds)
    </Text>
    <Text style={styles.textStyle}>
      Protocols Used: {item['Number of Protocols Used']}
    </Text>
    <Text style={styles.textStyle}>

```

```

    Servers: {item['Number of Servers']}
</Text>
<Text style={styles.textStyle}>
    Average Packet Size: {' '}
    {Number(item['Device Avarage Packet Size'].toFixed(1))} (Bytes)
</Text>
<Text style={styles.textStyle}>
    Data Flow Rate: {Number(item['Device Flow Rate'].toFixed(1))}{' '}
    (Bytes/Second)
</Text>
<Text style={styles.textStyle}>
    Data Flow Volume: {item['Device Total Flow Volume']} (Bytes)
</Text>
<View
    style={{
        flex: 1,
        flexDirection: 'row',
        justifyContent: 'space-evenly',
        marginTop: 100,
    }}>
<TouchableOpacity
    style={{
        backgroundColor: unknownbgColor,
        marginTop: 50,
        width: 190,

```

```

        height: 50,
        alignSelf: 'center',
        margin: 20,
        borderWidth: 1,
        borderColor: '#303841',
        justifyContent: 'center',
        alignItems: 'center',
    }}
    onPress={() => toggleSwitch()}>
<Text
    style={{
        fontSize: 20,
        fontFamily: 'Menlo',
    }}>
    Unknown Device
</Text>
</TouchableOpacity>
<TouchableOpacity
    style={{
        backgroundColor: knownbgColor,
        marginTop: 50,
        width: 190,
        height: 50,
        alignSelf: 'center',
        margin: 20,

```



```

        borderWidth: 1,
        borderColor: '#303841',
        justifyContent: 'center',
        alignItems: 'center',
    }}
    onPress={() => toggleSwitch()}>
    <Text
        style={{
            fontSize: 20,
            fontFamily: 'Menlo',
        }}>
        Known Device
    </Text>
    </TouchableOpacity>
    </View>
    </View>
);
}
}

```

```

const styles = StyleSheet.create({
    viewStyle: {},
    textStyle: {
        fontSize: 20,
        color: 'black',
    }
});

```

```
padding: 10,  
borderRadius: 5,  
fontFamily: 'Menlo',  
},  
buttonStyle: {  
padding: 10,  
backgroundColor: '#303841',  
width: 200,  
alignSelf: 'center',  
marginTop: 35,  
},  
});
```

```
export default DeviceInfo;
```