UCLouvain

École polytechnique de Louvain

IoT security analysis: Kunbus Revolution Pi connect+

Authors: **Antonio Rafaele, Nestor Eduardo Ramos Perez**
Supervisor: **Ramin Sadre**
Readers: **Olivier Bonaventure, Gorby Kabasele**
Academic year 2019–2020
Master [120] in Cybersecurity

Abstract

The purpose of this thesis is to study and analyze the limitations, in terms of security and performance, of the new Kunbus modules (i.e., Revolution Pi connect+ and DIO) based on Raspberry Pi Compute Module 3. For this purpose, we used vulnerability scanning softwares as well as offensive hacking tools such as, Metasploit, LOIC (Low Orbit Ion Cannon), Nikto, e.t.c. The performances of the Revolution Pi connect+ were measured based on the CPU load and the network load during an attack.
For that matter, we started by a thorough analysis of the device from a hardware and software perspective. This allowed us to have a better understanding of the possible limitations and issues with such a device.
Following that, we were able to set the different methods we wanted to use in order to achieve the objectives of our study. Those methods reach different aspects of the device, from hardware with the hard disk image cloning and communications to the DIO module, to software with the implementation of common services used on this kind of device.
Our results showed that the Revolution Pi connect+ has some network and CPU limitations to perform its regular tasks during an attack. Indeed, the results depicted an important number of network packets being dropped as well as a high CPU usage depending on the strength of the attack. The results also demonstrated the robustness of the device with its communications towards the DIO module. In fact, during an attack there was no alteration in the regular workflow of the device, in regard to the communication between the Revolution Pi connect+ and the DIO module.

Acknowledgment

We would like to sincerely thank our supervisor Ramin Sadre for his professional guidance and feedback over this paper. This thesis will always remain special with the current pandemic situation. In that matter, Professor Ramin was in contact with us during the whole quarantine. We would also like to thank Nicolas Omer for reading the paper and giving his feedback. We would also like to thank our readers Olivier Bonaventure and Gorby Kabasele.

Last by not least, we would like to thank Université Libre de Bruxelles (ULB), Université Catholique de Louvain (UCL), Université de Namur (UNamur), Royal military school and Haute-école Bruxelles-Brabant (HE2B) for organising and helping in this master degree.

Contents

Chapter 1

Introduction

1.1 The Internet of Things

The Internet of Things (IoT) is an emerging phenomenon where objects - *things* are connected to the internet. In this concept, many physical real-world objects will have a virtual entity and will connect and interact to benefit humanity [1]. Many applications of IoT are being implemented and used: smart homes, smart shopping, smart offices, smart cities and many more. This is caused by the commodity that those devices bring. They help along with comfort for humans (e.g., light bulbs connected to your smartphone, smart locks which open the door when you get close enough to the door, e.t.c). IoT has a strong effect on waste management. Thanks to the seamless integration of light bulbs, heat and air conditioning, a lot of money can be saved by adapting those services to our real consumption.
IoT is also used in smart cities [2] helping to reduce expenses and enhance life quality. The IoT supports a large population with basic services as data collecting meteorological weather stations, traffic sensors, e.t.c [3].

1.2 Background and objectives of the thesis

Nowadays, the number of IoT objects has grown exponentially. According to [4], in 2016 the number of estimated IoT devices was 6.4 billion (without including tablets, smartphones and computers). The expectation for 2020 is around 30 billion IoT devices. The increase of objects connected to the internet has brought with it an increasing in the possible attacks. By conception, these IoT devices are more vulnerable than a standard computer because of the low computational capabilities of the device. Indeed with lower capabilities, it is more difficult and less efficient

to implement security mechanisms (i.e., application firewalls, Intrusion Detection Systems (IDS), encryption, e.t.c.). In fact, starting from the design of IoT, security was not a primary issue. The questions about security in IoT raised in 2016 with the *Mirai Botnet attack*. A botnet is a network of internet-connected things that get infected with malware and then are controlled by a *Master* who will feed them with tasks to perform. All of this without the real owner of the IoT device knowing what his device is doing in background.
The *Mirai Botnet* was not the first attack on IoT but it was the first wide-scale attack. This attack infected more than 600.000 IoT devices. [5]

In this thesis we will focus on an IoT device: the *Revolution Pi connect+* produced by Kunbus corporation. It is an IoT based on a Raspberry Pi Compute Module 3. As stated before, these kind of devices have very low computational power which makes it difficult to implement security mechanisms to protect confidentiality, integrity and availability. One of the biggest objectives of the thesis is to test out the capabilities of the Kunbus Revolution Pi connect+. We will focus on installing an Intrusion Detection System, such as Snort. We will study the limits of the device in preventing attacks by attacking it and observing at which point the device cannot perform its usual tasks. Moreover, we will measure the negative impact of security mechanisms implementation on the normal behaviour of the device.

1.3 Structure of the thesis

This paper is organized such that we begin with a literature study of the Internet of Things landscape to better understand the explosion of IoT. In Chapter 3 we will present in more details the Kunbus Revolution Pi connect+ device, from the hardware to software characteristics, the installation procedure and the main goal of such a device. In Chapter 4 we present Snort and MQTT, two softwares that will be used in our study. Chapter 5 outlines the different tests performed on the Revolution Pi connect+, from vulnerability scanning to more complex attacks (e.g., Denial Of Service attacks, e.t.c). The aim of this is to test the capacity of the Revolution Pi to detect some attacks and to handle them while performing usual tasks. In chapter 6 we display the results obtained and observed in Chapter 5. In Chapter 7, we provide an analysis of the observed results. Finally, in Chapter 8 we draw conclusions about the real limitations of the device concerning security as well as the robustness of the device to handle the communications between modules during an attack.

1.4 Contributions

Our main contribution is a thorough experimentation of the Kunbus Revolution Pi connect+ with security mechanisms implemented. This product developed by Kunbus is quite new, even if it is based on a Raspberry Pi Compute Module 3. Currently there are not others studies concerning the security capacities of the Revolution Pi connect+. Therefore, with the different tests and attacks performed on the device we will help security experts and companies to understand to what extend the Revolution Pi connect+ can implement security related tasks.

Chapter 2

Literature study

2.1 Internet of Things: definition

The Internet of Things paradigm was first introduced in 1998 by Kevin Ashton. The idea of embedding short-range mobile computers into gadgets and everyday items, enables new ways of communication between people and things but mostly between things and things. The impact of the rapid growth of IoT can already be perceived in many aspects of every-day life. The best example is smart homes where the house we live in can have many objects connected in order to simplify our every-day lives (e.g., smart locks, smart bulbs, smart heating systems, e.t.c). The definition of IoT is still unclear. It is reasonable to define the IoT as

> "Things having identities and virtual personalities operating in smart spaces using intelligent interfaces to connect and communicate within social, environmental, and user contexts'. A different definition, that puts the focus on the seamless integration, could be formulated as 'Interconnected objects having an active role in what might be called the Future Internet'. The semantic origin of the expression is composed by two words and concepts: 'Internet' and 'Thing', where 'Internet' can be defined as 'the world-wide network of interconnected computer networks, based on a standard communication protocol, the Internet suite (TCP/IP)', while 'Thing' is 'an object not precisely identifiable'. Therefore, semantically, 'Internet of Things' means 'a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols' "[6].

2.2 Internet of Things: architecture

IoT architecture is divided in multiple layers [7]: perception layer, network layer, middleware layer and application layer (see Figure 2.1).

Figure 2.1: IoT architecture layers

The perception layer is responsible for capturing data physically. The network layer allows a common media for communication. Middleware and application layers are responsible for data usage in applications.

2.3 IoT gateway

One of the great strength of Internet of Things is the connection it can create between people and objects, objects and objects with no barrier to information and communication. As defined by [8], the definition of IoT gateway has evolved a lot over time as the Internet of Things devices have improved significantly. The more global definition of an IoT system would be bridging or connecting two concepts being the Internet of Things and network. [9] specifies that it is able to support different types of sensor nodes, multiple communication protocols, both wireless or wired, and provide a set of unified information for the application or user, making these only responsible for data processing. It means that, despite data being exchanged in different formats and languages, they can still be understood and processed easily by other systems than the sender.

Security wise, it is also very interesting as the sensors and devices get connected to the IoT gateway which will in turn translate and process everything needed to send to the data center or the cloud. Thenceforth, less machines must be used in the outside world and the security can be more center focused on the gateway. As a result, it is very important for these devices to be reliable, dependable and to possess high security requirements.

2.3.1 Industrial Internet Of Things (IIoT)

IIoT concept is explained in article [10] as being an extension and use of IoT in industrial sectors and applications. It is strongly focused on machine-to-machine (M-2-M) communications, big data and machine learning. Furthermore, it enables industries and enterprises to have better efficiency and reliability in their operations. What sets them apart from IoT is the intersection of information technology (IT), which refers to the utilisation of a computer to store, retrieve, transmit and manipulate data or information, and operational technology (OT), which refers to hardware and software that detect or cause a change through direct monitoring and/or controlling of industrial equipment, assets, processes and events.
In fact, these concepts provide industries with greater system integration in terms of automation and optimization, as well as better visibility of the supply chain and logistics. The monitoring and control of physical infrastructures in industrial operations are made easier through the use of smart sensors and actuators, which is a component from a machine responsible for moving and controlling a mechanism or system, as well as remote access and control. These type of devices are crucial to use in situations related to connected ecosystems or environments. Besides, they open space for many possibilities with their consistent capturing and transmitting of data among smart devices and machines.

2.4 Internet of Things: security issues

As declared above, the Internet of Things is a promising concept but as every new popular technology/concept new threats emerge. We will enumerate and explain the different security issues associated with IoT.

In [7], IoT devices are described as being physically present inside our homes as well as outside, they are opened to anyone most of the time, letting communications between devices and users open to everyone. As a result of resource constraints, applying cryptographic algorithms is practically unfeasible. This gave birth to a new research area *Lightweight cryptography* [11]. According to NIST [12], "Lightweight cryptography is a sub-field of cryptography that aims to provide solutions tailored for resource-constrained devices(...)". Wireless Sensor Networks (WSN), Wireless Body Area Network (WBAN), IoT, etc are some examples of the possible applicationq of lightweight cryptography.

Lightweight cryptography primitives include lightweight stream cipher, lightweight block cipher and lightweight hash functions [7].

Security was not a priority when IoT development began. Therefore, there are some vulnerabilities that have been found by the community [13]:

- Use of default passwords
- Passwords stored in plain text
- Communications are not encrypted
- Outdated firmware and no encryption
- Direct interface to internal network but also connected to the internet increasing the risk of an attack

It is known that hackers are exploiting those vulnerabilities. A well-known vulnerability scanner for IoT devices is *Autosploit* [14]. The aim of *Autosploit* is to automate the exploitation of hosts. Targets are collected via *Shodan* [1]. In this way IoT devices are identified and a *candidate* list of *victims* is fetched. Afterwards, the exploit part of the script tries a series of *Metasploit*[2] modules in order to exploit a vulnerability and once it has been done, the attacker drops the *payload* to interact with the compromised system.

Each layer of the IoT architecture (see Figure 2.1) faces multiple threats and some threats are present for several layers. For instance in the **perception layer**,

[1] *Shodan* is the world's first search engine for internet-connected devices

[2] https://www.metasploit.com/

a high threat is the *physical capture* of the device since it is in plain view to anyone. Another high threat is the *clone node* of the device because the hardware structure is simple and highly available to purchase hence one can replace the original device with a copy of that node which will be transmitting and receiving data according to the needs of the intruder.
The **network layer** is the most vulnerable one since everything is connected to the internet. One existing threat is *routing attack* since IoT communications are opened to anyone. Once the intruder gains access to the network, data forwarding can be done easily. Another threat possible is *Denial of Service attack* where the device can be made unavailable for the user by exhausting the resources (e.g., buffers). Finally, there is a *corruption* threat where an intruder can tamper data by using a narrow-band jammer with a 1000 times weaker signal to corrupt the reception of packets. The intruder can alter the packets by introducing some bits and therefore the transmitted data will be corrupted.
Concerning the **middleware layer**, a high risk attack named *Data attack* can be done by altering the packet header. The attacker acts as a *man-in-the-middle* by passively monitoring the messages.
Finally, one possible attack for the **application layer** is *privacy leak*. The use of default passwords by the users helps considerably the attacker to brute force the authentication system. Another possible threat is *malicious code*. Indeed the attacker can append malicious code on the software application.

2.5 Secure architecture for IoT

Various secure architecture related works have been done due to the explosion of IoT. In [15], the authors proposed a secure and efficient authentication and authorization architecture (SEA) for the healthcare environment IoT devices. Secure management scheme is crucial to accomplish this. The proposed architecture is distributed hence mitigation (at least reduction) of Denial of Service (DOS) attack is possible.

Object Security Architecture was proposed in [16], they describe an architecture for end-to-end security in the IoT. It is based on the concept of object security that relates security with the application payload. The architecture includes Authorization Servers that provide clients with Access Secrets that enable them to request resources from Constrained Application Protocol (CoAP) nodes. The nodes reply with the requested resources which are signed and encrypted.

In [17], authors presented a secure architecture for 6LoWPAN[3] using compressed

[3]Standing for: IPv6 Low power Wireless Personal Area Networks

IPsec. Their aim is to extend 6LoWPAN such that IPsec communication with IPv6 nodes is possible. Therefore using IPsec enables true end-to-end security and removes the need for a trustworthy gateway. Mainly, they provide End-to-End (E2E) secure communication between IP enabled sensor networks and the internet.

2.6 Related work

In [18], the authors discussed and deployed Snort (well-known Intrusion Detection System, will be presented in Chapter 4) on a Raspberry Pi. At the end, they concluded that a Raspberry Pi is suitable to perform intrusion detection in an IoT environment. Nevertheless, their experiments took place in a home network thus experiments on a bigger network are needed to have a better perception of the true limits of a Raspberry Pi.
Notice that the authors used an older version of the Raspberry Pi (i.e., Raspberry Pi 2 Model B). Their research is related to ours because they were also testing the limits of a Raspberry-like device for security purposes.

In [19], the authors also deployed an IDS on a Raspberry Pi in order to measure the performances of a Raspberry Pi and its capability to run an IDS. From their experiments, they draw the conclusion that a Raspberry Pi can be used as an IDS depending on the users throughput demands. Indeed in their experiments, they observed a throughput drop which exceeded 30%.
In this study they mainly focused on the performances of running an IDS on a Raspberry Pi. However, in contrast with our study, they did not test the negative impacts of running an IDS on the normal operations of the device.

In [20], three types of honeypots are implemented in IoT environments. Honeypots are a security mechanism set to lure hackers into monitored environments that simulate real services. First, they implemented a medium-high interaction honeypot which simulated a specific series of router UPnP [4] services. Second, they used the actual IoT device firmware to help the honeypot respond to indistinguishable malicious requests.
Their experiments were successful because they could catch many unknown malicious attacks.
This work was an implementation of a security mechanism, such as honeypots, in an IoT environment. Therefore, it is related to our own work because it is a security mechanism that is often only available on regular servers but they implemented it in an IoT environment.

[4]Networking protocol standing for: Universal Plug and Play

Nonetheless, they did not study the negative impact of such an implementation on the normal operations and performances in their environment.

In [21], the authors studied the security challenges and limitations in IoT technology. They listed several causes of the security challenges that IoT faces. Some are: lack of skills, architecture challenge of IoT, data storage in IoT devices, lower power and capacity, weak security testing, e.t.c.
The authors mentioned that it is a challenge to implement security measures on IoT devices since they lack storage capacity, CPU and power.
They did not analyze in depth why it is challenging to implement security measures. That is why our work is convenient to analyze thoroughly why security mechanisms are a challenge to implement on IoT devices and also how these implementations impact (positively or negatively) the usual tasks of those devices.

Chapter 3

KUNBUS Revolution Pi

3.1 About KUNBUS company

KUNBUS[1] is a German-based company specialised in fieldbus, which is an industrial computer network protocols used for real-time distributed control, and industrial Ethernet solutions in the automation sector. It was founded in 2008 with the aim of providing innovative and economical products to the steadily growing market of industrial networks. They do not only provide embedded communication modules, but also gateways as well as a variety of network diagnostics and monitoring devices.

3.2 The Revolution Pi

The KUNBUS company has brought with the launching of the Revolution Pi an open, modular and small computer based on the established Raspberry Pi[2]. It is a very flexible device which can be used for a wide variety of purposes. It meets the EN 61131-2 standard, also known as IEC 61131-2, which is part of a series of standards on industrial control equipment as described in [22] and is equipped with Raspberry Pi Compute Module. The base module can be extended seamlessly using right appropriate digital and analog I/O modules as well as fieldbus gateways. For instance, in this paper the base module is the Revolution Pi Connect+ and the extended module is the Revolution Pi DIO. Following that, a whole small network could be made with multiple modules and each having their own purpose.

Due to its open platform and open-source concept, the Revolution Pi offers to the users maximum freedom when implementing automation projects. For a

[1] https://www.kunbus.com/
[2] https://www.raspberrypi.org/

while now, systems were rarely made open-source since secrecy always looked like the best practice. This can apply with old concepts as GSM systems for example which kept everything hidden. The latter has improved by giving only to scientist or researcher their mechanism in details. Recently more and more open-sources systems and softwares were made and a lot of debates emerged around closeness or openness. A lot of researchers were concerned on that matter and Andreas Schmidt made a very interesting doctoral thesis named *Secrecy versus openness*[23] which highlights both interests and problems of secrecy and open-source. Finally, open-source solutions has proven to be more attractive and beneficial for every company using it. As an example, Microsoft[3] were famous for being against the open source solution but have recently changed and started embracing the approach as mentioned [24].

Moreover, it offers a great variety of software ready for use which will be discussed later in further details. It allows a full root access which implies obstacle free programming and implementation of customized programs. Individual applications can be programmed via Python, C or Node-RED.
The system gives a lot of possibilities opening its services to any kind of users. It has the balance between the basis of a Raspberry Pi that can be a useful tool for learning and the power to create a great professional project. The gateway possibilities should not be underestimate as it allows the connection between devices from different networks. The latter empower the system strength and utility. Indeed, the system could also react or use the information exchanged between the multiple different network devices.

The Revolution Pi has established Raspbian operating system from Raspberry Pi, including the drivers for the expansion modules. However, all these utilities do not implicitly cover all security aspects and might give space for vulnerabilities.

3.3 KUNBUS modules

The main module used in this paper is the Revolution Pi Connect+ which can be seen in Figure 3.1 and the second attached module, which is directly connected to the main one through a physical Bridge, is the Revolution Pi DIO module which can be seen in Figure 3.2. The Connect+ has multiple useful functionalities as it will be discussed later and will be used accordingly. In the other hand, the DIO is, if it stays untouched, a simple and empty module doing nothing special. To unleash the later potential, it is important to configure it as the owner desired.

[3]https://www.microsoft.com/

Once it is done, the device can be found as a powerful, reliable and effective tool with a lot of different assets. For the purpose of the thesis, the later will be used as a little gadget allowing a LED to switch on and off repeatedly after a predefined amount of time.

Figure 3.1: Revolution pi Connect+ module

3.3.1 Connect+ module specifics

The KUNBUS Revolution Pi Connect+ has been built with the idea of allowing users a maximum freedom of design. It has two Ethernet sockets enabling the device to simultaneously connect with two different network which could, as an example, let data being transfer from a network to another. It is powered by the Raspberry Pi Compute Module 3+, which means quad-core power with 1.2GHz and 1GB RAM as well as up to 16GB of storage. In addition to the two Ethernet sockets, it also uses two USB sockets, one micro HDMI socket for connecting a video monitor and one micro USB socket. Finally, as an operating system, the company pre-installed a specially adapted Raspbian operating system based on Debian 9, which is equipped with a real-time patch. The use of Raspbian ensures that most of the applications developed for the Raspberry Pi can also be used on the Revolution Pi. Also, the real-time patch allows to maintain a high level of control over the priorities of the tasks that the scheduler manages. In fact, the scheduler, which controls the execution of the tasks managed by the operating system, can be configured extensively to avoid the delay caused by network access and other I/O access operations.

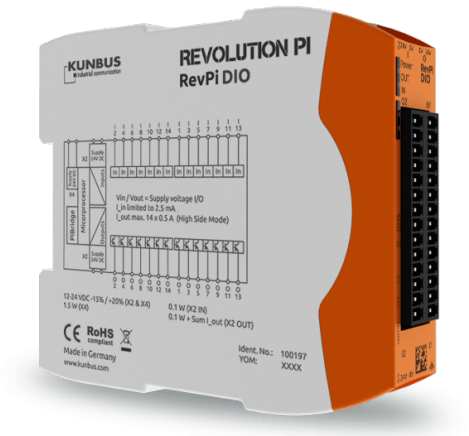

Figure 3.2: Revolution Pi DIO module

Finally, one of the main goals of the Connect+ is to act as an IIoT gateway. It is indeed for this reason that the top-hat rail housing of the Revolution Pi Connect+ is fitted with a robust 24 V industrial hardware that complies with the EN 61131-2 standard. It also has an Electrostatic discharge (ESD) of 4 kV/8 kV to protect the circuits and both Surge/Burst and Electromagnetic Interference tests.

3.3.2 DIO module specifics

The Revolution Pi DIO is a digital I/O module complying with EN61131-2 standard, used to expand the Revolution Pi system. It contains 28 pin I/O connectors with 14 for digital inputs and 14 for digital outputs. It is connected with the Connect+ through a PiBridge logic circuit which is a 20 pin system connector working as a bus coupler for stackable modules allowing communications between both devices. This last matter results in the need of DIO own power supply connector.
Inputs and outputs are cyclically exchanged via PiBridge with the Revolution Pi Connect+ central processing image using the PiControl driver as referenced by [25].
Additionally, this module does nothing by itself, it is empty. It needs to be configured and setup by an user. Once this is done, it becomes a reliable and powerful tool.

3.3.3 Connection between Revolution Pi Connect+ and DIO

As mentioned previously, PiBridge is a device allowing both Connect+ and DIO to communicate. For further details, as mentioned by [26], the connection is not directly made. Indeed, in the beginning every module starts in a "waiting for configuration" state. Only the Connect+ reacts through RS485 communication by exchanging status data, detecting the module type and comparing it to the type registered in a configuration file. RS485 is a standard defining the electrical characteristics of drivers and receivers for use in serial communications systems. If everything complies, the DIO gets its specific configuration data including its address. If multiple modules are connected, this configuration step is repeated one-by-one. On the contrary, if the configuration file does not match with the DIO configuration, an error occurs indicated by LED signals. If everything goes smoothly, the data transfer is switched to the highest speed (115,200 bps) and everything is ready and the cyclical data exchange of process data starts.

3.4 Software characteristics

In terms of software, there are two main options to take into consideration. The first one being the proposed software from the organisation whereas the second one being the possibility of implementing the user own software. Amongst all the software offered following the first option, TeamViewer[4], KUNBUS Cloud and PiCtory are the main interesting ones security wise.

Teamviewer is a well word-widely known remote desktop software. The device can be connected to the software using the Internet. It is made possible by means of a very secure and user-friendly TeamViewer technology to access the Revolution Pi webserver via a browser window.

On the other hand, KUNBUS Cloud enables the Revolution Pi Connect+ to provide a cloud service. A software agent installed on the Revolution Pi establishes the highest level of security and at the same time user-friendliness. As with the previous software, it can be also be configured via internet. KUNBUS cloud can be booked as a service or installed on a user device.

Finally, PiCtory is a configuration software installed into the device. To have access, the user must connect to the device using the latest IP address and, once on the software page, login with the provided username and password. The software

[4]https://www.teamviewer.com/en/

impact is proportional to the architecture size which implies an important security implementation. It is a very powerful tool letting the user setup or modify any parameter of any Revolution Pi modules connected. It has a whole interface letting the user configure any modules within his likings.

3.5 Revolution Pi modules particularities

In the eyes of any computer science expert, the Revolution Pi modules are depicted as a simple condense computer or simply as a Raspberry Pi. In fact, the module is based on the latter, however it is much more powerful. Not only it has multiple modules that can be used for a lot of different purposes, but it also has multiple interesting build-in functionalities. Revolution PiModIO as an example is an integrated python library which, as specified in [27], provides various functions to access the hardware of the Revolution Pi with the device owner Python program. It is possible then to make fast IO-Checks over the network, upload the user control program, start it at system start and transfer all IO data via MQTT. Or simply write data from any Python system to the process image. It consists of two main components:

- Revolution PiPyLoad: It is running as a service on the Revolution Pi in the background. It starts the transferred Python program for the control and monitors it. The outputs of the program are written into a log file.
- Revolution PiPyControl: It is a graphical Interface for controlling, monitoring and transferring Python Programs. It allows all tasks to be performed over the network with the Revolution PiPyLoad. It runs on Windows, Mac or Linux, Python is the only requirement. This program allows to create the user own Revolution Pi systems in a list to allow quick Access.

3.6 Installation procedure

Every step from the installation to the configuration will be explained in this section.

3.6.1 Connect+ installation

For the installation of the Revolution Pi Connect+, the first step consisted in connecting it to a power supply. To do so, the device has an X4 plug with 24V supply, ground and functional earth entry. Using the X4 plug, these entries must be linked with a wire to the power supply using the same entries. Another type of

plug is implemented, the X2 plug, though it will not be used in this case on the device.

Once it is done, the device can be powered on, however it will be totally isolated. Depending on the user purpose, the second step resided on linking the device into one or two different networks. At this state, the device can be accessible through its IP address where the user, after login, can configure multiple options. Amongst these, softwares like PiCtory can be launched and others can be activated as another way to communicate with the Revolution Pi (e.g., Teamviewer).

Following the simple implementation represented in Figure 3.3, the Revolution Pi is both accessible locally and remotely. In fact, by adding rules into the internet modem, it is possible to allow any outside connection looking for a given port, to directly communicate with the Revolution Pi. This might lead to important flaws in the system if the device is not well secured.

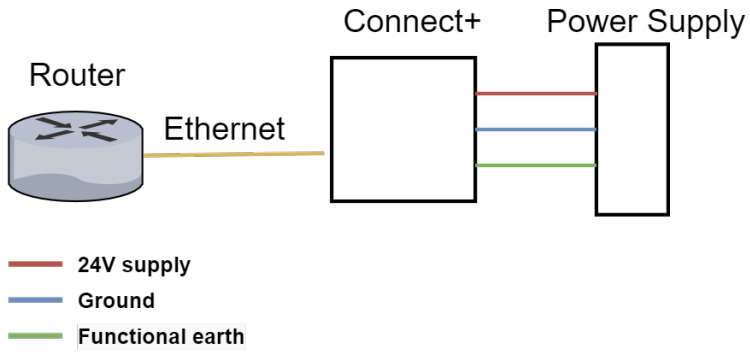

Figure 3.3: Simple representation of the Revolution Pi installation with respectively: a router, the Revolution Pi and the power supply.

3.6.2 DIO and LED installations

The connection between the DIO module and the LED is shown in Figure 3.4.

DIO installation

The first step consisted in powering on the module. For that purpose, the device was connected, in a very similar manner than the Connect+, as it also has an X4 plug. Once everything is setup, both devices were still not interacting with each other and that is where the PiBridge comes into action. By connecting both modules with the latter, communication was made.

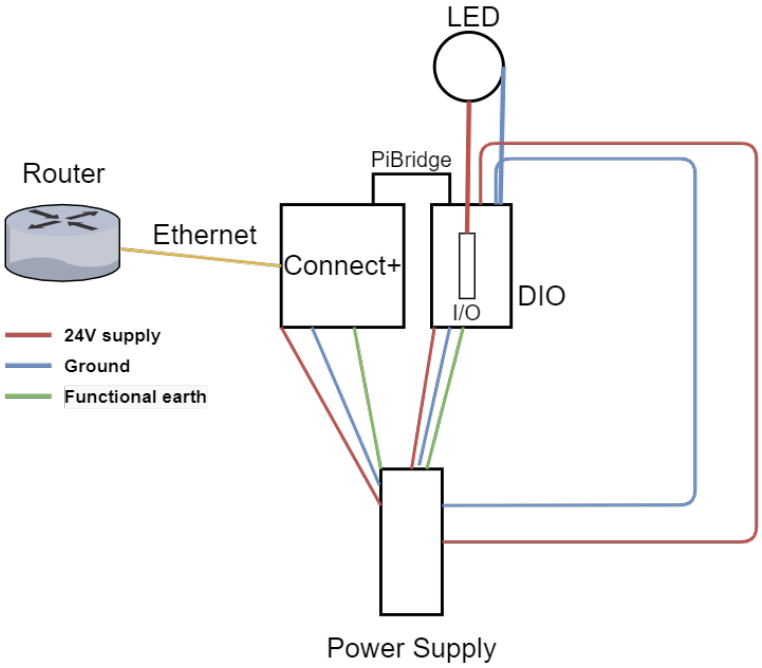

Figure 3.4: Simple representation of the DIO and LED installation.

Finally, After all of the precedent steps are done, there is still a problem as the DIO will still not be ready to be used yet. Indeed, it must first be configured by using PiCtory. Therefore, the successive procedures must be be followed:

- Connect to Connect+ module and Log in
- Launch PiCtory
- Add DIO module to the left of Connect+
- Click on *File* and *Save as Start-Config*
- Click on *Tools* and *Reset driver*

As a result, DIO is fully recognized and configured and can be employ safely.

LED installation

For experimentation purposes, as it will be describe in a succeeding chapter, a LED will be directly connected to the DIO module with the objective being to be able to switch it on/off. This last functionality is operated by a direct connection with the DIO output.

In that matter, it is first needed to power on the output by wiring up the X2 plug, located on top of the device, to a power supply as it also needs its own electrical source. Afterward, X2 plug Ground entry power supply must also be connected to the LED. For the switch functionality implementation, a wire must be connected between both DIO output and LED.

Finally, the configuration is completed in the same way as with DIO configuration. Moreover, with the help of PiCtory, it is possible to set a default value to the switch to turn the LED on by default or off.

Chapter 4

Background

4.1 Snort

Snort is an open-source network intrusion detection system (IDS) and intrusion prevention system (IPS) created in 1998 by Martin Roesch, founder and former CTO of Sourcefire. Currently, Snort is developed by Cisco since they purchased Sourcefire in 2013 [28].

In this thesis we will mainly focus on the IDS part of Snort software. An intrusion detection is of the utmost importance since detecting quickly an intruder may reduce the risk of further damage and penetration. Intrusion detection is based on the assumption that the intruder *behaves* differently than a normal user. Of course, the segregation between the intruder and a normal user is not simple. To have a better understanding of the distinction between a normal user and an intruder, see Figure 4.1. We observe a clear overlap between a normal user behaviour and an intruder. Therefore, intrusion detection systems have a difficult task to distinguish a normal user from an intruder. For this reason, based on the interpretation of the overlap that an intrusion detection system does, the number of *false-positives* could arise meaning that normal users are identified as intruders. On the contrary, the number of *false negatives* could increase meaning that the IDS identified wrongly an intruder as being a normal user. It is important to mention in that regard that the *false negatives* are way more dangerous than *false-positives*.

In [30], the authors identify two approaches to intrusion detection:

- **Statistical anomaly detection**: "Involves the collection of data relating to the behavior of legitimate users over a period of time"[30].
- **Rule-based detection**: "Involves an attempt to define a set of rules that can be used to decide that a given behavior is that of an intruder"[30].

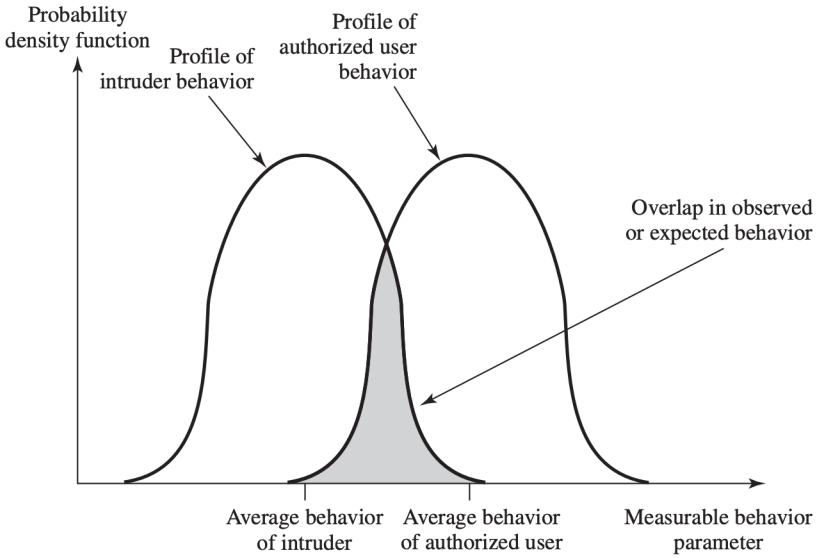

Figure 4.1: Profiles of behaviour of normal and intruders users [29]

Snort is a *rule-based* intrusion detection system. Indeed, based on the rules configured, a packet can generate different actions:

- *alert*: an alert is generated and then the packet is logged.
- *log*: log the packet.
- *pass*: ignore the packet.
- *drop*: drop and log the packet.
- *reject*: block the packet, log it, and then send a TCP reset if the protocol is TCP or an ICMP port unreachable message if the protocol is UDP.
- *sdrop*: drop the packet without logging it.

Another important feature of an intrusion detection system is its location. Thus, we have two types of *IDS*: *Network-based intrusion detection systems* and *Host-based intrusion detection systems*. The first type of *IDS* is placed directly on a physical network directly on a network device (e.g., routers, switches, ..). The second type of *IDS* is placed directly on a host system to only protect a specific host.
Snort belongs to the first category (i.e., *Network-based intrusion detection systems*). It is used to find suspicious traffic in the network the *IDS* is placed (see Figure 4.2).

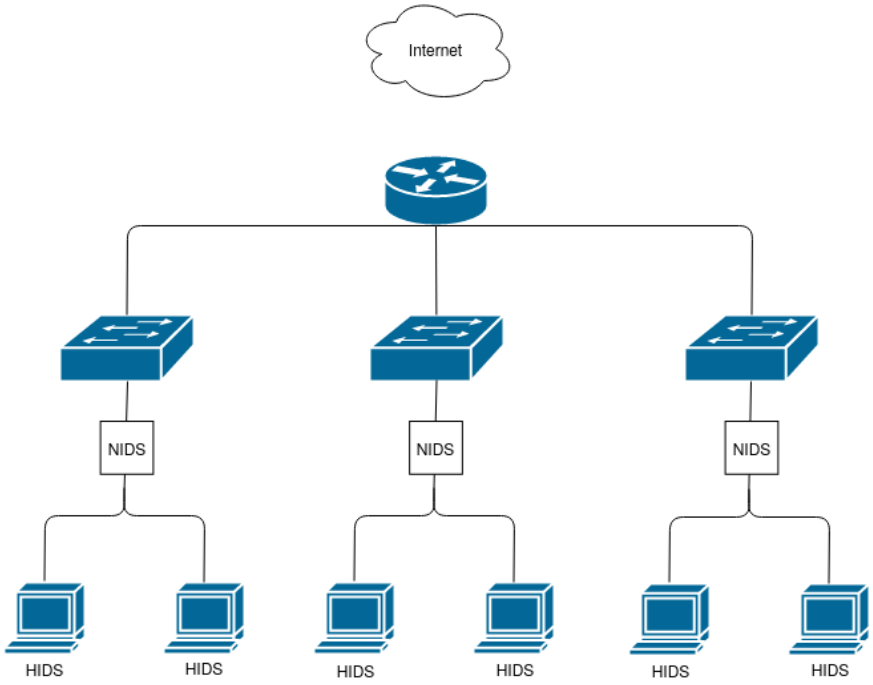

Figure 4.2: Network-based IDS and Host-based IDS[31]

In the Chapter 5, we will describe in detail the installation and configuration of Snort on the KUNBUS Revolution Pi connect+. Finally, the rule sets that were used in the thesis will be presented and their use will be justified.

4.2 MQTT

4.2.1 Overview

As stated in [32], the Message Queue Telemetry Protocol (MQTT) is a protocol used for communications within an IoT environment that functions on top of TCP. It was created by IBM as a M2M (i.e., Machine-to-Machine) lightweight protocol. MQTT is a messaging protocol for low-resources devices, it is useful for connections with remote locations where the network bandwidth is limited. The protocol is based on a publisher-subscriber architecture (see 4.2.2 section) where the *subscribers* do not request updates from the *publisher* but instead *subscribe* to a certain topic and "as soon as" the *publisher* pushes new data the *subscribers* receive it. This mechanism frees up the network load by reducing the number of messages needed between the *publisher* and the *subscribers*.

Furthermore, MQTT handles reliability through three types of reliability mechanisms called Quality of Service (QoS). The *publisher* and the *subscriber* reach an agreement, in the connection process, on the guarantee of delivering a message. The three QoS levels are:

- 0 - at most once: the *publisher* simply sends a message without receiving any acknowledgment. Message loss can occur.
- 1 - at least once: the *publisher* sends a message and receives an acknowledgment message for it. Notice that in case of a loss of the acknowledgment message, the *publisher* will send its initial message again. Duplication of messages possible.
- 2 - exactly once: the *publisher* is sure the message he sends will be received because there is a *handshake* between him and the receiver.

The higher the QoS level is, the higher the load of the communications will be.

4.2.2 Architecture

In the previous section we presented a quick overview of the MQTT protocol without going into details. Here under, MQTT architecture will be displayed and discussed more precisely.

As shown in Figure 4.3, the publisher and the subscriber are present as stated in the overview. However, we can note the appearance of a new object to the architecture, the *Broker*. The main job of the *Broker* is to handle the communications between the *publishers* and the *subscribers*. In other words the *publishers* and the *subscribers* do not communicate directly to each other. This architecture is a client-server structure where the clients are the *publishers* and *subscribers* and the server is the *Broker*. For instance in the scenario at Figure 4.3, the *Humidity sensor* acts as a *publisher* and both *On-Premise Analytics* and *Cloud Analytics* act as *subscribers*.
The *Humidity sensor* connects first to the *Broker*, identifies himself as a publisher and creates a topic humidity. On the other hand, the *subscribers* connect to the *Broker*, identify themselves as subscribers and subscribe to a topic (in this case humidity). Henceforward, every time the *Humidity sensor* publishes data, the *Broker* will push the new data to all *subscribers* subscribed to the topic *humidity*.

Clients (i.e., publishers and subscribers) can perform the following actions [34]:

- opening the Network connection to the *Broker*.
- publishing Application Messages that other Clients (i.e., *subscribers*) might be interested in.

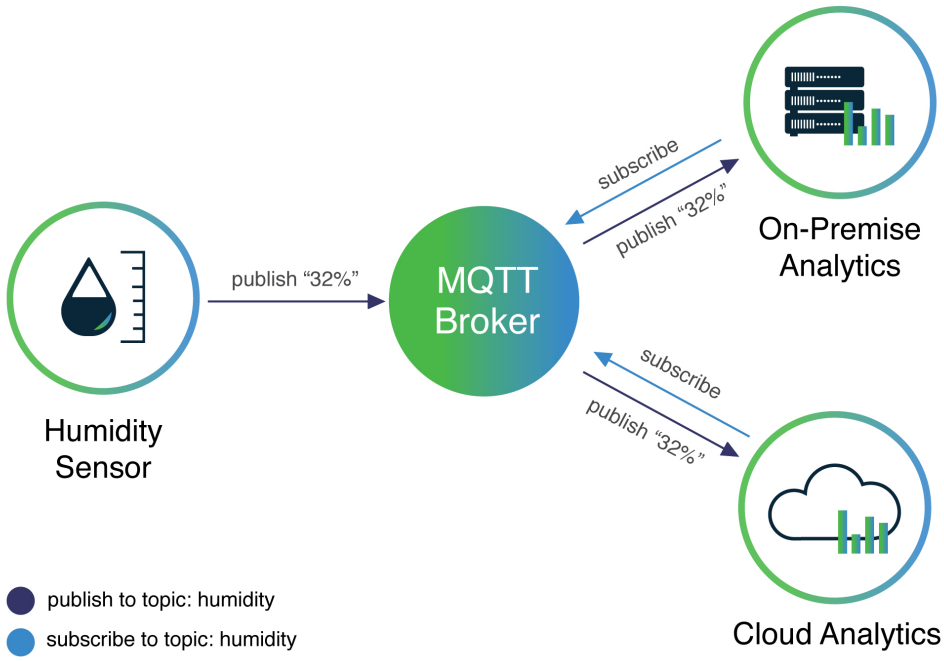

Figure 4.3: MQTT architecture[33]

- subscribing to requested Application Messages that it is interested in receiving (i.e., topics).
- unsubscribing from topics.
- closing the Network Connection to the *Broker*.

On the other side, the *Broker* can perform the following actions [34]:

- accepting Network Connections from Clients.
- accepting Application Messages published by Clients (i.e., *publishers*).
- handling Subscribe and Unsubscribe requests from Clients.
- forwarding Application Messages that match Client (i.e., *subscribers*) subscriptions.
- closing the Network Connection from the Client.

4.2.3 Security overview

MQTT provides few security mechanisms which are either not configured or provided with default settings. In the official documentation provided by oasis, there is a section called *Security* [35] in which all the security mechanisms are explained and some security recommendations are given. For instance, the authentication of clients by the *Broker* is handled via the CONNECT packet containing the *username* and *password* fields provided by the clients. On the other hand, authentication of the *Broker* by the clients has only been supported since April 3rd, 2019 with the update to MQTT v5.0. The introduced enhanced authentication boosts the basic authentication to include challenge-response style authentication which might involve the exchange of AUTH packets between the client and the *Broker* after the CONNECT and before the CONNACK packets.

Chapter 5

Methodology

5.1 Overview

As stated in the objectives section, the aim of this work is to study the limits of the Revolution Pi device in terms of security and performance. The prime question of interest is: would the Revolution Pi connect+ still handle its usual tasks while being stressed by attacks and by security mechanisms implemented?
In order to bring answers to this question, several tests were made on the device. In this methodology chapter, we describe step by step the experiments performed to achieve the objectives of this thesis.

To this extent, Section 5.2 is dedicated to vulnerability scanning of the device. In Section 5.3 we describe Snort implementation as well as the different rule sets used. Section 5.4 explains MQTT implementation and the objective behind it. Section 5.5 describes a test performed with the DIO module and a connected output LED. Section 5.6 depicts the main attack performed on the device, a DoS attack. Finally, Section 5.7 refers to another kind of attack executed where the complete image of the device could be cloned.

5.2 Vulnerability scanning

5.2.1 Background

Secure Shell (SSH)

The secure shell services, also shorten as SSH services, is a cryptographic network protocol for operating network services securely over an unsecured network. It can

for example allow a secure remote login over an insecure network. It is based on three major components as described in [36]:

- The Transport Layer Protocol: provides server authentication, confidentiality, and integrity. It is typically run over TCP/IP connection.
- The User Authentication Protocol: authenticates the client-side user to the server. It is run over the Transport Layer Protocol
- The Connection Protocol: multiplexes the encrypted tunnel into several logical channels. It is run over the User Authentication Protocol.

Once a secure transport layer connection has been established, the client sends a service request which is followed by a second one after the user had completely authenticated itself and so, the authentication has been made.

Following that, SSH is widely used protocol and very useful in this case to interact with the Revolution Pi. Nevertheless, a lot of tools exists to crack users password using SSH connection. In fact, SSH does not verify the number of time a user tried to connect in a small amount of time which leads to possible dictionary or brute-force attacks. To do so, hydra or Nmap [1] are two very well known tools which could lead to cracking users passwords as for example displayed in Figure 5.1 and 5.2. Indeed, it is very common for users to have weak passwords as for

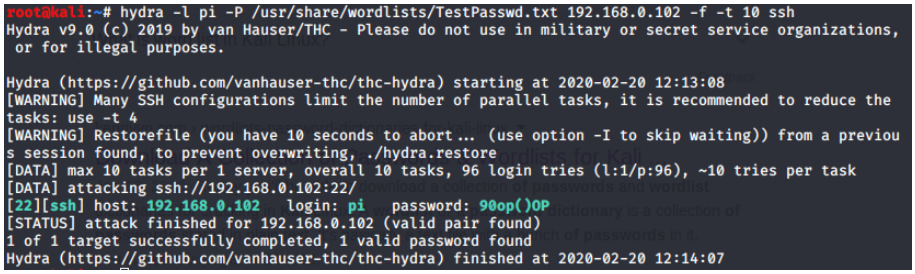

Figure 5.1: Cracked password using Hydra

example passwords of less than 8 characters or passwords easily breakable using a word list combined with a set of rules. For that purpose, it is crucial to set long passwords with random characters to lower the chance of being hacked and even more necessary as the device default password is very weak.

[1] https://nmap.org/

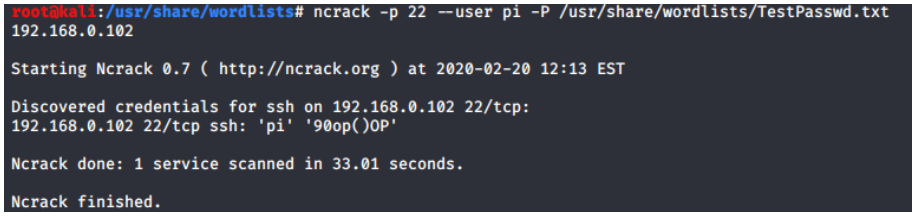

Figure 5.2: Cracked password using ncrack

HTTP and HTTPS

On the one hand, HTTP stands for Hypertext Transfer Protocol and is a application-level stateless and generic protocol defining how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. it does not provide any channel integrity protection, confidentiality, or secure host identification as stated in [37]. The lack of security makes this protocol nowadays unused as HTTPS is simply a more secure and upgrade of HTTP.

On the other hand, HTTPS stands for Hypertext Transfer Protocol Secure and compared to HTTP, encrypts all the communication protocols using The Transport Layer Security(TLS) which primary goal, as highlighted by [38], is to provide privacy and data integrity between two communicating applications. The layer is composed of two layers:

- TLS Record Protocol: ensures the connection privacy, using symmetrical key for data encryption, and reliability, implementing a message integrity check.
- TLS Handshake Protocol: used for encapsulation of various higher-level protocols. It allows the client and the server to authenticate themselves and choose the encryption algorithm and encryption key to have a secure communication between both entities.

Following that, there is no interest to have an open http port even by default. Inversely, it is a bad practice security wise as a username and password could be retrieved by an attacker. To do so, the attacker could sniff the traffic by, for example, being a man in the middle and retrieve the hash code of the user. This latest can be seen in Figure 5.3.

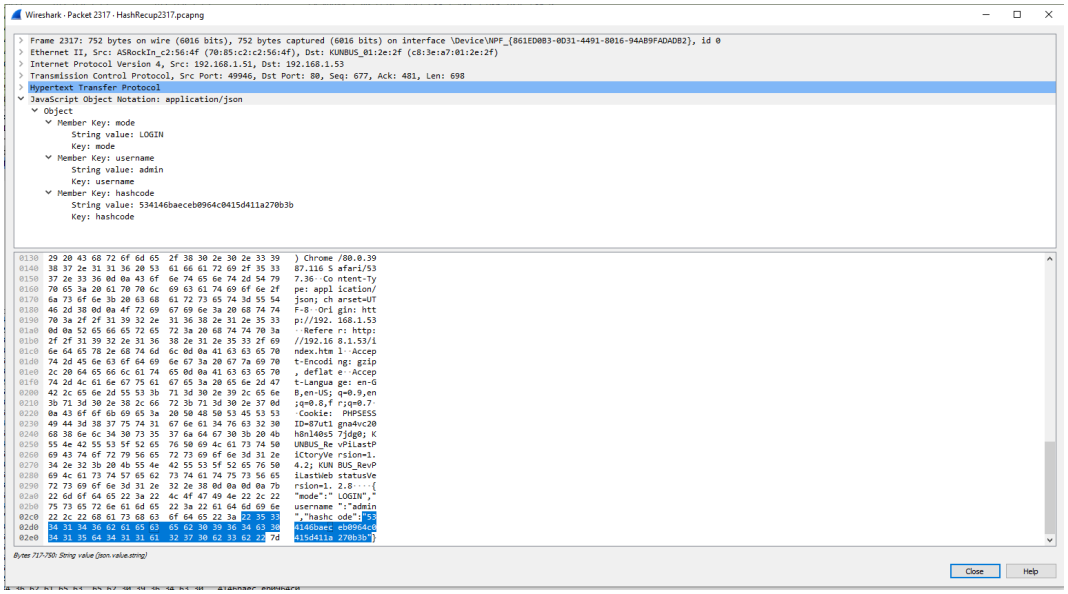

Figure 5.3: Wireshark used for sniffing network traffic between client and Revolution Pi

Ultimately, the service should not be used as it has no interests and can create flaws in the system.

5.2.2 Nmap scanning

After the installation of the device, multiple services are accepted and opened to communicate with it. All these services can be discovered using Nmap which is a free and open source tool for vulnerability scanning and network discovery. It is a very useful tool with multiple utilities as it can identify what devices are running on a system, discover hosts that are available and the services they offer, find open ports and detect security risks. Nmap results list all open services, as shown in Figure 5.4, and with these, some issues can be noticed.

5.2.3 Vulnerabilities scanning with Nessus Pro

The first thing we did was to run a vulnerability scan against the Revolution Pi connect+. For that matter we used Nessus Pro [2] from *Tenable solution*. Tenable was named the leader in vulnerability risk management by Forrester[39].
We installed Nessus pro (just a 7-days trial) on a Mac OS machine. The configuration of the scan was simple in our case since we only wanted to scan one host

[2]https://fr.tenable.com/products/nessus

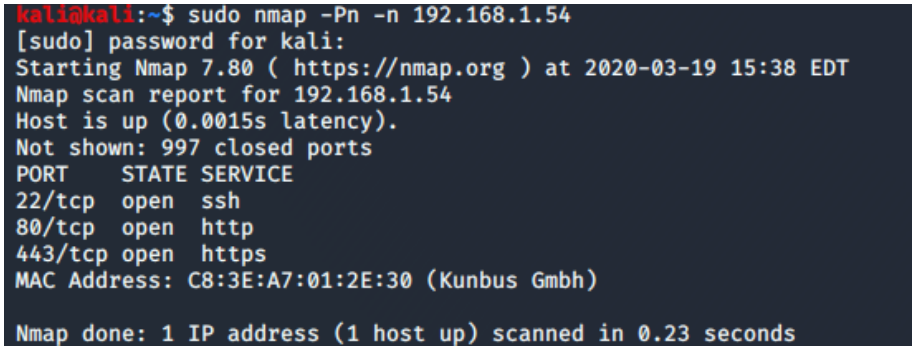

Figure 5.4: Nmap scan

(i.e., the Revolution Pi connect+). The vulnerabilities raised by Nessus are of three types: *critical*, *high* and *medium*. The *medium* is a special one since it means Nessus could not check some controls and a manual verification has to be done. The idea behind starting with a Nessus scan was to see known vulnerabilities from the Revolution Pi connect+ before exploring manually other possible vulnerabilities.

5.3 Snort implementation

5.3.1 Snort installation

For this thesis we installed Snort 2.9.15.1. It is the latest most stable version of Snort, Cisco have published a new version of Snort (i.e., Snort 3.0) but it is a Beta. Therefore, we choose the latest and more stable version *Snort 2.9.15.1*.

Note that Snort is not recommended to be run on Raspberry Pi's systems. In fact, the official hardware requirement states that minimum 4 GB of RAM are required[40]. As stated above in the KUNBUS Revolution Pi presentation, the Revolution Pi connect+ has only 1 GB of RAM. Hence, theoretically running Snort would not be possible. However in practice running Snort in the Revolution Pi connect+ is possible because the number of running services in our case is limited (e.g., ssh, PiCtory and Procon-web-Iot).

For the installation, because we use the Revolution Pi connect+ in command-line mode (i.e., no GUI), the process is more arduous as we need to install the dependencies separately. In the listing 5.1 we listed all the commands used in the installation process. We will not go into detail the meaning of each command since it is not relevant for the main goal of the thesis.

Listing 5.1: Commands for the installation of Snort

```
1 $ sudo wget https://www.snort.org/downloads/
```

```
   snort/daq-2.0.6.tar.gz
$ sudo tar -xvzf daq-2.0.6.tar.gz
$ cd daq-2.0.6/
$ ./configure
$ sudo apt install flex
$ ./configure
$ sudo apt install bison
$ ./configure
$ make
$ make install
$ wget https://www.snort.org/downloads/snort/
   snort-2.9.15.1.tar.gz
$ sudo tar -xvzf snort-2.9.15.1.tar.gz
$ cd snort-2.9.15.1/
$ sudo ./configure --enable-sourcefire
$ make
$ make install
$ ldconfig
$ sudo ln -s /usr/local/bin/snort /usr/sbin/
   snort
```

5.3.2 Snort configuration

Once the installation process is done, Snort creates a folder *etc/snort/* where all the configuration files , pre-processors and rules are stored.

The configuration process is pretty simple, at least for the part that interested us in the scope of this thesis. The most important configuration file is *snort.conf*. This file is mainly divided in 5 sections:

- **Network variables**: rules locations, IP addresses definitions, e.t.c.
- **Decoder configuration**: the decoder determines which underlying protocols are used in the packet its decoding.
- **Preprocessor configuration**: preprocessors are used to perform certain actions before a packet is processed by the main detection engine.
- **Output modules configuration**: those modules handles how data is logged.
- **Rules configuration**: rules files (i.e., *.rules*) are included here.

In our case, we modified the *network variables* with our actual values (e.g., home-net IP address, ports to activate,...). The second section we modified was *rules*

configuration: by default the *snort.conf* file has a lot of included rules, some are not necessary for our purpose and as we will discussed in the next section, it is crucial to not overload the Revolution Pi connect+ with unneeded rules.

5.3.3 Rules sets

As specified above, the Snort configuration file needs the path to our rules folder. By default Snort comes with basics rules thus we wanted to see the available rules sets. On the official Snort website[41], we can download different rules sets: *community* rules, *registered* rules and *subscription* rules. A payment is required for the last rules set, hence, we used the *community* rules and *registered* ones.

Initially we started by using the registered rules since they are more complete (see Figure 5.5). To do so we configured the paths variables to point to the registered rules we downloaded from the official website[41]. Then, when we started Snort in the *Console* and *alert* mode, in the initialisation process Snort was killed by the Operating system. This Snort crash will be studied in more details in the analysis section 7.
In consequence we downloaded the community rules set, which is drastically smaller than the registered rules. Thus we changed the rule path in the *snort.conf* file to point to a new folder *community-rules*. Unsurprisingly, Snort did not crash and could operate correctly.
Those rules are updated by the community regularly and a rule pool could be configured in order to receive the latest updated rules by the community security experts. However, because of the power and memory limitation of the Revolution Pi, we did not configure a rule pool.

Figure 5.5: registered rules

5.4 MQTT implementation

MQTT was implemented for our thesis using the *Eclipse Paho project* which is an open-source client implementations of MQTT. Again remember the goal of this thesis is to study the limits of the KUNBUS Revolution Pi connect+ regarding security. Therefore the idea behind implementing MQTT on the KUNBUS Revolution Pi connect+ was to test the device using a common configuration as MQTT.

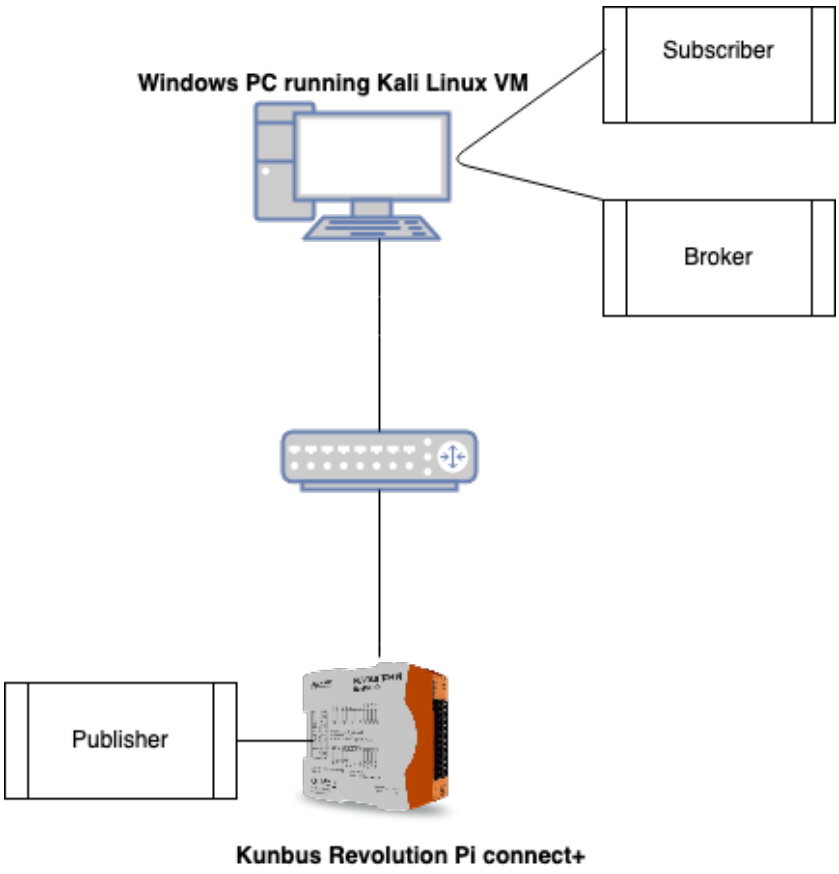

Figure 5.6: MQTT set up

As explained before in 4.2.2, we need two MQTT clients (i.e., *publisher* and *subscriber*) and one MQTT *Broker*. The *publisher* is installed on the Revolution Pi connect+, the *subscriber* and *Broker* are installed on a Kali Virtual Machine. Our implementation is shown in Figure 5.6/

Installing MQTT clients

The installation of the *publisher* on the Revolution Pi connect+ is done with the following command: *sudo apt-get install mosquitto-clients*. The same command is used on the Kali Virtual Machine for the installation of the *subscriber*.

Installing MQTT Broker

The *Broker* is also installed on the Kali Virtual Machine with the command: *sudo apt-get install mosquitto*. As explained in section 4.2, we needed to test the Revolution Pi connect+ used as a MQTT *publisher* and communicating with a MQTT *Broker* (installed on a virtual machine) which will forward the published data to the *subscribers* registered to the topic being pushed.

On the *publisher* side (i.e., Revolution Pi connect+), we wrote a simple python script using the *Eclipse Paho project* API. In the script displayed in Appendix A.1, the Revolution Pi connect+ publishes random numbers (i.e., from 0 to 100) every 0.5 seconds.
Some things to notice: first the connection to the *Broker* at line.11 in A.1, second at line.17 the publishing of the message (i.e., the random number generated at line.16). We observe that the first argument of the *publish* function is the topic we are publishing new data to (i.e., *test/randomInt*), the second argument is the actual payload (i.e., the random number) and the third argument is the *qos* (i.e., quality of service). The details regarding the *qos* were already discussed in the MQTT section 4.2. It is also interesting to observe the line.10 where we set an username and a password (in this case the password was not set), it is used by the *Broker* too identify *publishers* and *subscribers*.
The code was then executed using Python 3.7.3 with the command *python3 publisher.py*.

On the *Broker* and *subscriber* side, they are both running on a Kali virtual machine, there is no script running them but instead we used two command lines in two separate terminals windows.
On the *Broker* terminal we executed *sudo mosquitto -V* to run it. Before running the *Broker*, the configuration file *mosquitto.conf* had to be modified in order to configured the *Broker* properly. The details of this file will be omitted in this thesis as the configuration was simple. On the *subscriber* terminal we executed *sudo mosquitto_sub -t test/randomInt -h 192.168.1.26* and the *mosquitto_sub* client was ready to receive new published data from the *Broker*.

5.5 Running the DIO with a LED in the output

The Revolution Pi DIO is an additional module which can be directly connected to the Connect+ via PiBridge. It has multiple functionalities as discussed previously (in Chapter 3, Section 3.3.2), including the most important one for this purpose being the output configuration. In fact, for this experiment, a LED has been connected to the DIO and a script has been created to make the LED switch off and on every 0.5 seconds as illustrated in appendix A.2.

The goal of this experiment is very similar to the precedent one with MQTT but slightly different. In fact, for this approach, everything is connected via wires whereas MQTT is through the network. Following this approach, a DoS attack will be executed on the Connect+ allowing an examination of the hardware behavior when it is put under pressure.

5.6 DoS Attack

A denial-of-service (DoS) attack is, as the name states, a method to make a machine or network resource unavailable to its intended users by disrupting services of a host connected to the Internet for an undetermined amount of time. The difference between a DoS and a distributed denial of server (DDoS) is that the first uses one computer and one Internet to flood a targeted system or resource where the other uses multiple computers and Internet connections. A lot of different types of DoS attacks exists and they typically consist on flooding the targeted machine or resource with irrelevant requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled. As stated by [42], it is often capable of bringing a well-crafted and protected service to its knee which is the principal interest of using it against the Connect+. In fact, the aim of attacking KUNBUS device with this sort of attack is to investigate about its behavior while being pressured by a high load of requests.

Several types of tools have been used to simulate a DoS attack. The most successful one were Metasploit for synflood and LOIC[3] for TCP and UDP flood (Low Orbit Ion Cannon) DoS attacks. These tools were used to store in a pcap file the attack behavior which was then used with TCPreplay allowing to control the attack power and pressure over the Connect+.

[3]https://sourceforge.net/projects/loic/

5.6.1 Metasploit DoS Attack tools

Metasploit is a well known, widely distributed and open source tool written in Ruby which can be used for a lot of different causes. In fact, it contains a lot of different utilities as network scanning, payload generation, obfuscation and most importantly for this testing, DoS attack modules. Amongst all the different implemented constituent, one is of great interest considering our purpose: synflood.

Synflood

TCP synflood attacks are well described in [43] by being a method aiming at the CPU memory. Whenever a connection through a server is made by a client, a three way handshake protocol is established before any actual data transfer. A SYN packet is send in the first place by the client to initiate the handshake. The server then replies to the client with an acknowledge packet. Finally, the client sends a SYN ACK packet to establish a successful connection.

In a synflood attack, as the name suggest, the three-way-handshake will not be completely established as the client will never send the last SYN ACK packet. In more details, when a connection initiated, as the server receives the first SYN packet from the client it will store in memory that the client wants to establish a connection. As the memory is of limited size, if an enormous amount of clients tries to connect and leave the connection open and thus filling up the memory, the server is most likely going to crash or have problems by not having the power and space to accept other connections.
To execute this attack with Metasploit the following steps were performed:

- Execute Metasploit. It can be done with Kali by executing `msfconsole` into a terminal.
- Use synflood module: `use auxiliary/dos/tcp/synflood`.
- Configure the DoS attack and mainly the RHOST, RPORT with the command `set`. To see all options, use command `show options`.
- Execute the attack with `run`.

An image illustration has been displayed in Figure 5.7.

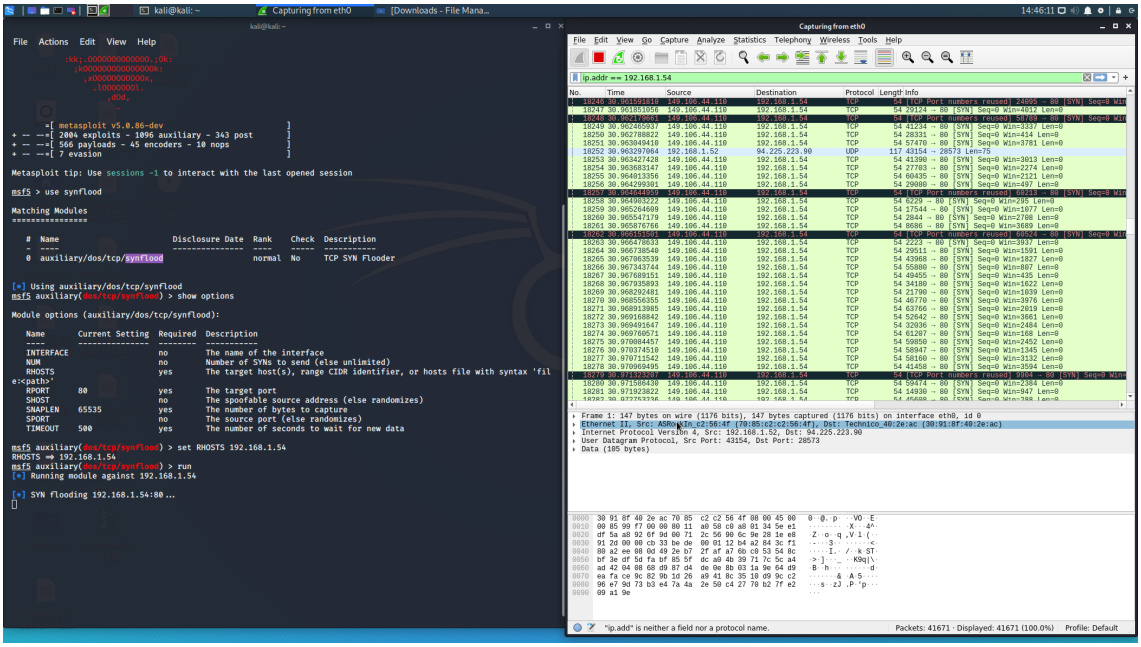

Figure 5.7: Synflood Metasploit module

5.6.2 LOIC DoS Attack

Low Orbit Ion Cannon (LOIC) is an open source software developed by Praetox Technologies for research purpose. It is used for network stress testing and can perform both denial of service and distributed denial-of-service attacks. It has the utility to flood target systems with junk TCP, UDP and HTTP GET requests. Both TCP and UDP attack modes send string message packet that can be personalised whereas the HTTP flood mode sends an endless volley of GET requests.
The succeeding steps can be followed to execute this tool:

- Disable Windows Security as it recognised the tool as being malicious where it is not. In that matter, it would be more secure to execute LOIC in a Virtual Machine.
- Configure all the parameters as IP address or URL of the target, the attack mode where UDP, TCP and HTTP can be chosen, the number of threads to create more requests and attack speed.
- Finally, execute by clicking on *IMMA CHARGIN MAH LAZER*.

An image illustration has been displayed in Figure 5.8.

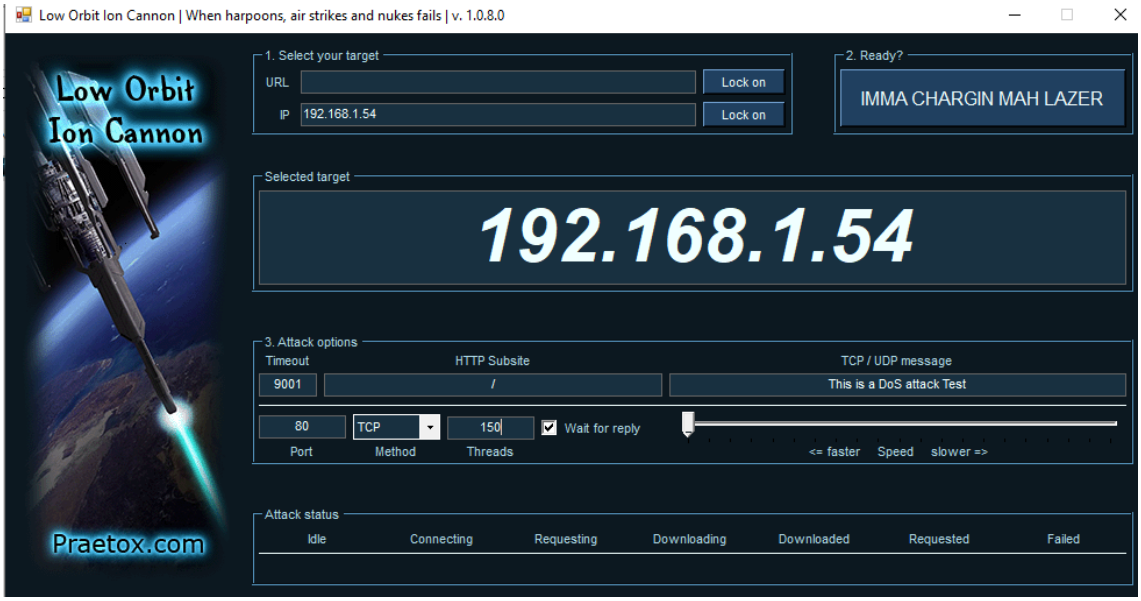

Figure 5.8: LOIC software

5.6.3 TCPReplay

TCPReplay [44] is a UNIX operating systems tool for editing and replaying network traffic which was previously captured. It allows to classify the traffic as client or server and replay it back onto the network.

The purpose of using TCPreplay in this case is to create a real and precise simulation of a DoS attack. In fact, with all the helpful options proposed by TCPreplay, the traffic control functionality was very interesting to test the device limits. Moreover, it looked like a true and well simulation of an DoS could not have been possible without this tool as the bandwidth is limited in our implementation.

In that matter, the bandwidth restriction was found out thanks to `nload` command in a personal computer. In fact, the traffic speed of LOIC attack, in TCP mode with the highest speed was limited to approximately 90 megabytes per second which had no effect on the device as the local network bandwidth could not handle it. Each service discussed before was running well and no impact on performances was noticed on the device. Because of the machine having good resources such as 1 GB of RAM and of the bandwidth problem, it was not possible to oppress the device in a fair way and another method had to be found which is where TCPreplay comes into action.

To launch the tool, it is first needed to retrieve a pcap file containing the traffic over the device. To do so, one of the precedent described attack is used with the Connect+ as target. At the same time, any tool with the functionality of saving the traffic into pcap file, as for example Wireshark, is used. After a certain amount of time, the pcap is saved and sent directly to the Connect+ via secure copy. The file should not be too big as the device has a limited memory resource.
Finally, TCPreplay is launched as follows:

```
1    $ sudo tcpreplay -i eth1 -M 10 --loop 20 storedAttack.
       pcap
```

with `eth1` being the network interface, `-M 10` the number of megabytes per second which is in this case 10, `--loop 20` the number of loop to replicate the traffic in the pcap file and `storedAttack.pcap` the pcap file.

5.7 Disk image copying

During our research for vulnerabilities on the Revolution Pi connect+, we glanced through the KUNBUS Forum [45] and found out an interesting post about security. Indeed, the post [46] involves a security mechanism that is implemented but with no information on how to use it and how it works.
The security mechanism is the crypto chip *ATECC508A* from Microchip. As defined on the crypto chip website [47], the crypto chip is a secure element with advanced Elliptic Curve Cryptography (ECC) capabilities. With ECDH and ECDSA being built right in, the *ATECC508A* is ideal for the rapidly growing IoT market by easily supplying the full range of security such as confidentiality, data integrity, and authentication.

Nevertheless, KUNBUS organization is very conservative and the secrecy prevails concerning the crypto chip. Indeed on another post [48], the KUNBUS moderator "volker" answers to someone asking about the crypto chip. The moderator explains that:

> "The crypto chip is neither part of the open source concept nor is it free for use to our customers. There is no guarantee from KUNBUS that future devices may have production closed registers or important information stored into open registers. The chip is exclusively reserved for KUNBUS and customers who will contract to use this chip for their dedicated purpose in negotiation with KUNBUS [...]"[48].

In other words, the crypto chip is not part of the open-source project and no clarification is given on how this chip could be used for a project. This brings us

back to the first post [46] where another moderator, 'Mathias', clarifies that there is no encryption of the entire hard drive. Hence, someone with physical access to the Revolution Pi connect+ could possibly connect through the Micro-USB port and dump the entire image of the hard drive.

Consequently, we wanted to test how a malicious person with physical access could dump the hard drive image and have access to all files stored in the Revolution Pi connect+.

5.7.1 Software needs

To the extent of cloning the hard disk image, we need two programs: *RPIBoot* and *Win32 Disk Imager*. The first one enables the Revolution Pi connect+ eMMC as a mass storage device under windows which, in other words, allow the computer to recognised the device. The second one is used to create a raw image file of a removable device (i.e., the Revolution Pi connect+).

5.7.2 Image cloning procedure

To clone the image of the Revolution Pi connect+, the first step consisted in plugging the micro-USB port of the off powered device with a USB A port on a computer. Once it was done, the *RPIBoot* can be executed as displayed in Figure 5.9 and then power on the connect+. After pressing *Enter* on the keyboard, a window opened, shown in Figure 5.11, stating that the device has been recognized but must be repaired. The next step was simply to click on *Continue without check* and launch Win32Disk tool. Once the latter is opened, multiple choices must be made as the image name, its location and the connect+ location, which was displayed on the precedent recognition window or can be check in the folder *This PC* on windows. Finally, when everything was completed as shown in Figure 5.10, we clicked on *Read* which saved the image.

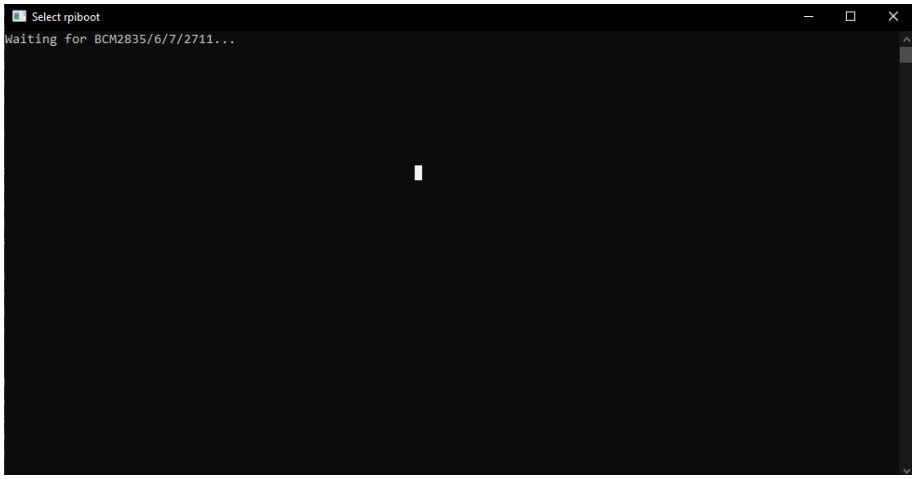

Figure 5.9: RPIBoot execution

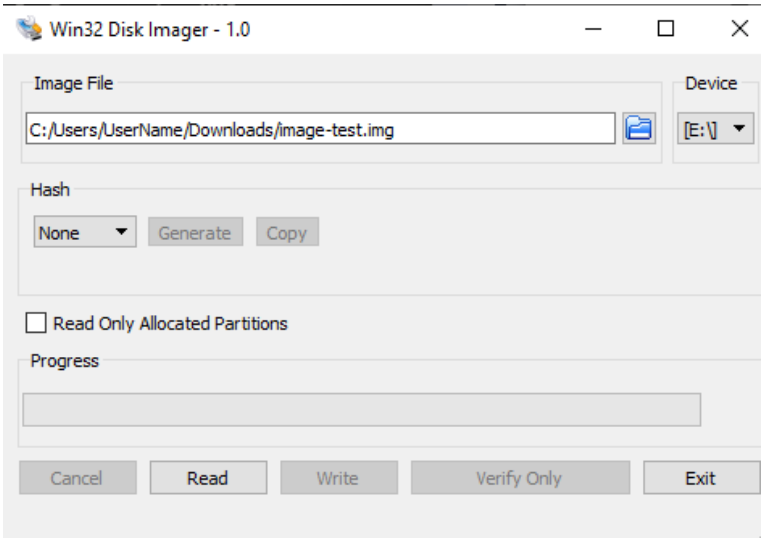

Figure 5.10: Win32DiskImage

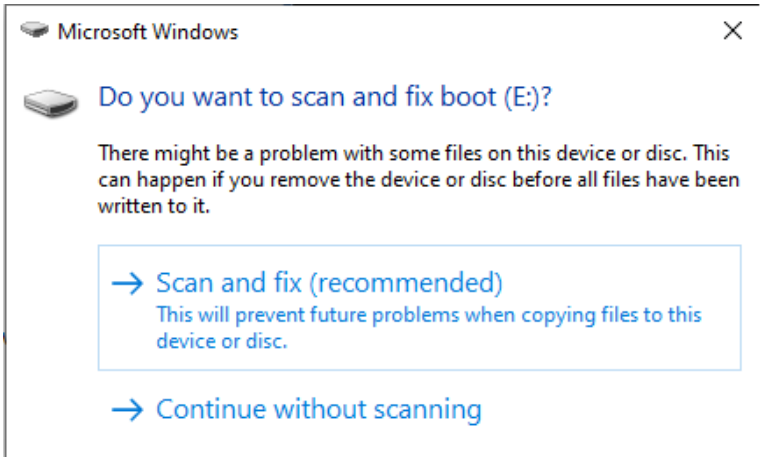

Figure 5.11: Window displayed when Connect+ is recognised by computer

5.7.3 Raw image mounting

In this scenario, the image was mounted using a Kali virtual machine, however it can be mounted using any operating system. To mount it, it is important with Linux to first know the offset of the partition as ".img" files are not images of a partition, but of a whole disk and start with a boot loader, which is a code booting an operating system, and a partition table. To do so, *fdisk* command was used as follow:

```
$ sudo fdisk -l <image_location>
```

Once the command executed and the partition found, a little computation must be done to find the offset: $Block\_size\_start \times 512$.
Finally, the mount command can be executed:

```
$ sudo mount -o loop,offset=<computed_offset> <
    image_location> <mounting_location>
```

Chapter 6

Results

6.1 Nessus Pro scan

The results of the scan are shown in Figure 6.1. On this Figure we observe: 4 *high*(i.e., orange color) vulnerabilities, 18 *mediums* (i.e., yellow color) and 66 *infos* (i.e., blue color). The *infos* alerts correspond to controls that were successfully passed by the Revolution Pi connect+.

On Figure 6.2, the *high* rated vulnerabilities are displayed. The two vulnerabilities are related and are caused by an old version of Apache running (i.e., Apache 2.4.25). We observe that Nessus alerts multiple vulnerabilities due to the outdated Apache version, when we clicked on the second one (i.e., Apache 2.4x $<$ 2.4.39 Multiple vulnerabilities) the list of *CVE* (i.e., Common Vulnerability and Exposures) identifiers is displayed: CVE-2017-3167, CVE-2017-3169, CVE-2017-7659, CVE-2017-7668 and CVE-2017-7679. In order to get details about a specific CVE we need to do a research on the NIST National Vulnerability Database[49].

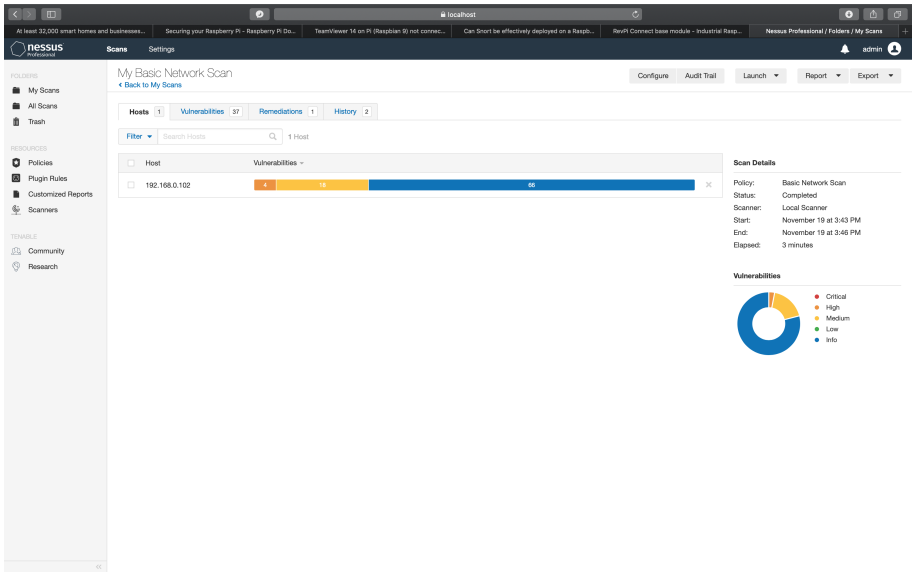

Figure 6.1: Output of Nessus Pro scan

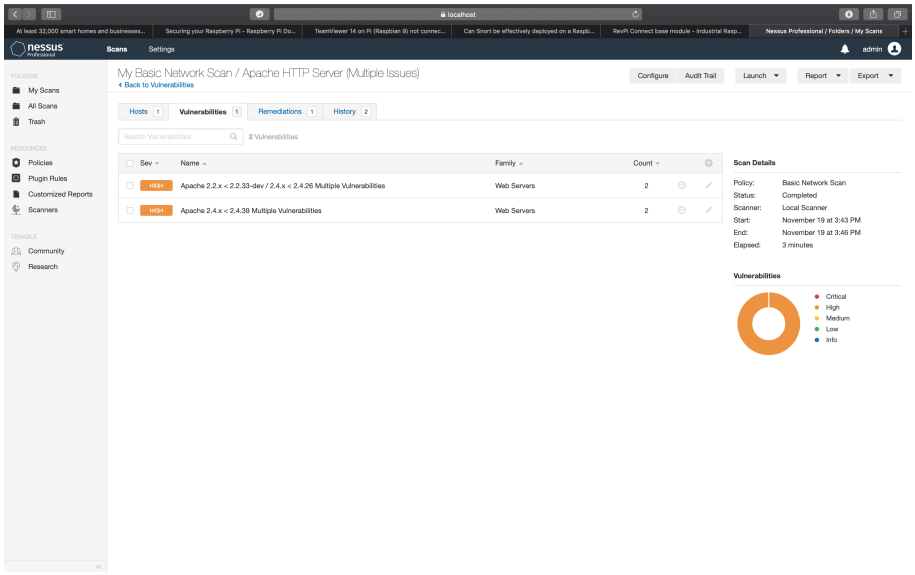

Figure 6.2: High vulnerabilities reported by Nessus

6.2 CPU usage during DOS attack

For the CPU usage measurement, we used the *top* command which displays the CPU usage and memory usage of all processes running.

The results are shown in Figure 6.3. The bars chart represents the CPU usage in percent depending on the DoS attack speed. We tested three different attacks speeds: 20 MBit/s, 60 MBit/s and 120 MBit/s.
The orange bars correspond to the CPU usage of the device, without running Snort. The blue bars represent the CPU usage of the device, with Snort running.
For each set of tests, we performed the attacks three times and we calculated the mean value of the CPU usage.

As we can see from the bars chart in Figure 6.3, there is a clear difference in the CPU usage percentage between the device running Snort or not. For instance, the attacks at 20 MBit/s loaded the CPU at 50.4% without Snort and at 65.6 % running Snort. At 60 MBit/s, the CPU load was 87.7% and 126.3%, without and with Snort respectively. Finally, with an attack speed of 120 MBit/s, the CPU load was 120.63% and 169.2% respectively.

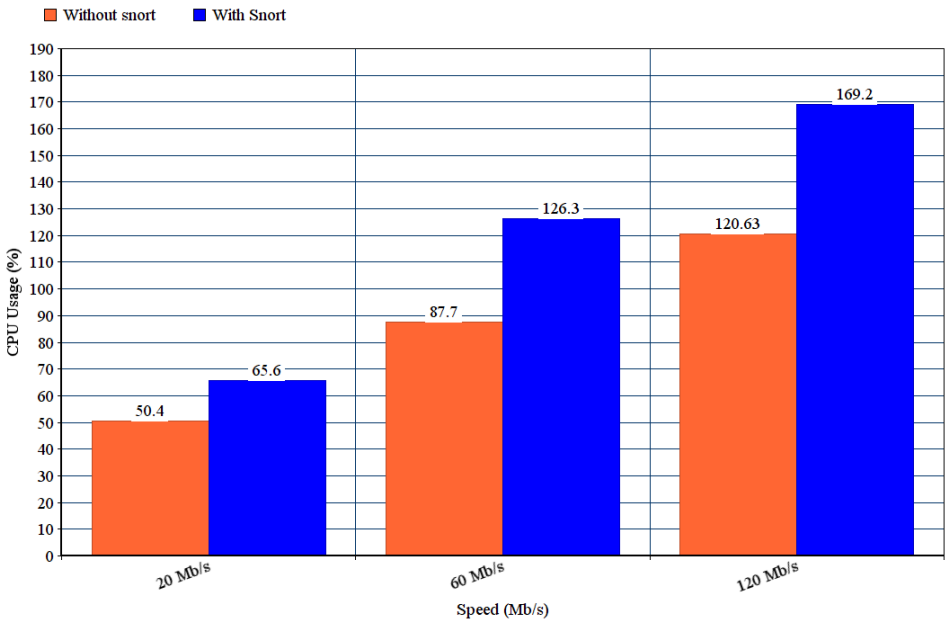

Figure 6.3: CPU usage during attack with different speeds

6.3 Network traffic

The network traffic was monitored using the *nload* command which displays the current network usage[50]. We will show two results: firstly the network usage before the attack and secondly the network usage during the attack.

Before the attack, network usage for the current incoming traffic was 816.00 Bit/s.

During the attack, the network usage observed was 78.76MBit/s. The total incoming traffic observed was 1.38GByte after 15 seconds since the attack started.

6.4 Snort limitations

The DOS attack performed (see section 5.6) had a crucial impact on Snort performances. When Snort is stopped, an output is displayed summarizing the number of packets received, analyzed, dropped, filtered, outstanding and injected.

For an attack speed of 1000MBit/s, Snort sniffed *454386* packets d and *141519* (i.e., 23.749%) were dropped.
Below 1000MBit/s, the percentage of dropped packets was insignificant (approximately 5%).

6.5 MQTT delays

Regarding MQTT, problems were encountered in the communications between the publisher and the subscriber. Indeed, during the DoS attack different delays were observed in the client side. In fact, for a publisher publishing each second, for a certain period of time each message is received normally. However, sometimes randomly a delay appears and varies from approximately 2 seconds to 25 seconds. It can reach higher amounts depending on the attack speed.
Also important to note that all packets are well received from the client side. What happened is that for a certain amount of time, the client does not receive anything. After that, it received a burst of packets in one go which are all the messages it should have collected during the void time.

The results of the observed delay are shown in Figure 6.4. At a speed of 5 MBit/s, zero delay was observed. At a speed of 10 MBit/s, a delay of 2 seconds appeared. At 15 MBit/s, the delay was up to 14 seconds. At 40 MBit/s, the delay observed was 16 seconds. Finally, with a speed of 80 MBit/s, the delay was up to

25 seconds.

Unequivocally, from the observed results in Figure 6.4, greater is the attack speed, higher is the delay and the burst of packets.

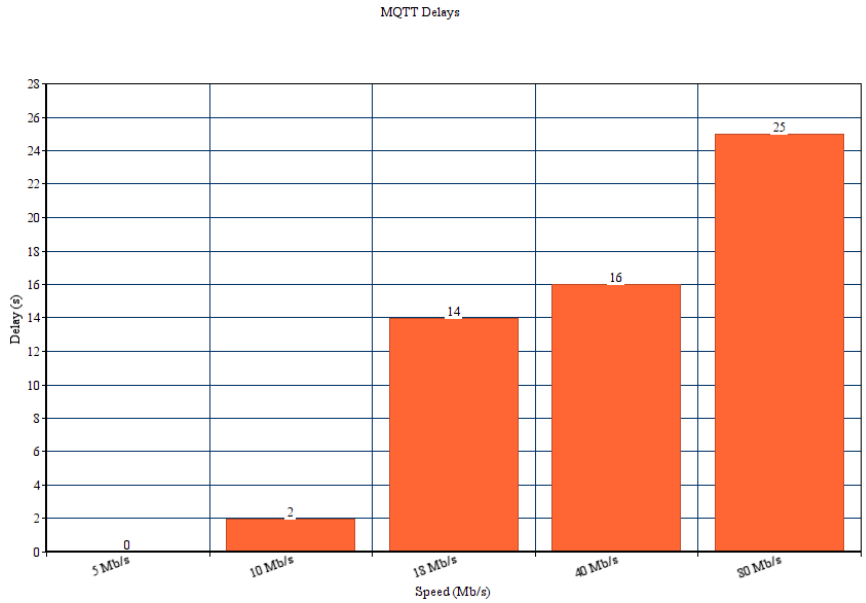

Figure 6.4: MQTT packets delay depending on the attack speed which is measured in megabytes per second (MBit/s.)

6.6 LED switching

Concerning the LED switching, while stressing the Revolution Pi connect+ by exercising a DoS attack, no issue was observed over the LED switching off and on. In fact, multiple switching speeds (i.e., from 20 MBit/s to 1000MBit/s) were tested and the script executed for this purpose was running normally without any internal errors. Furthermore, there was no delay or odd behavior found, everything was working correctly.

6.7 Revolution Pi connect+ image cloning

The results of the disk image cloning and mounting are shown in Figure 6.5 and 6.6. As we can see, the whole disk image was mounted on our Kali virtual machine and all files are displayed in clear-text.

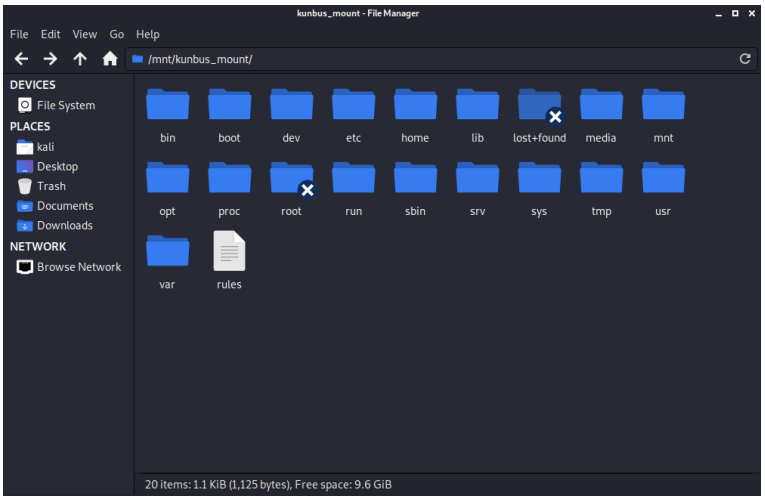

Figure 6.5: Revolution Pi connect+ image mounted

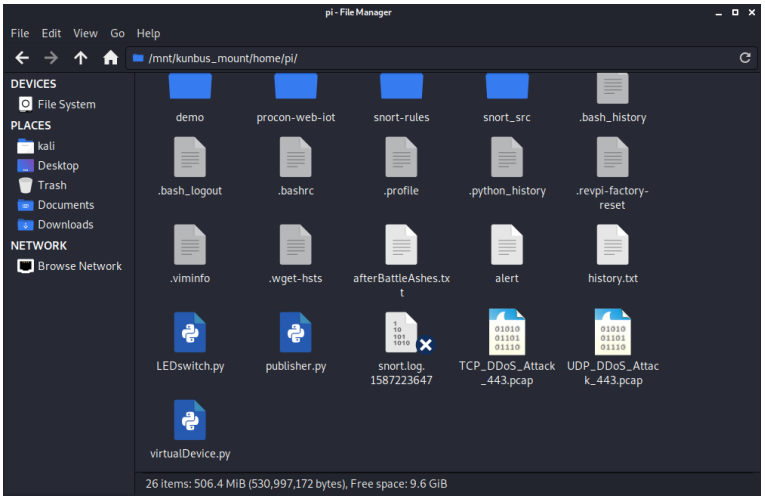

Figure 6.6: No-encrypted personal files in the mounted image

Chapter 7

Analysis of results

7.1 Vulnerabilities discovered by Nessus

The vulnerabilities found by Nessus were critical and most of them were discovered in 2017. It seemed odd that KUNBUS were distributing products (i.e., Revolution Pi connect+) with an outdated and vulnerable version of Apache. We investigated more about these issues and found out that in fact our Revolution Pi connect+ was not vulnerable any more to all the CVEs listed. Indeed, Debian manages packaging in a way to always have stable versions of their systems. For example, if Debian Stretch ships with Apache 2.4.25, a security issue is found and fixed in 2.4.39, Debian will take this security fix, and apply it to 2.4.25, and distribute the fixed 2.4.25 to its users[51].

Therefore, the Nessus scan thought there was a vulnerability with the version of Apache installed since it only checks the version number. Debian have a security tracker database where a specific CVE can be searched and it will tell if the vulnerability has been fixed or not on our current Debian version[52]. We searched the CVEs that were found by Nessus on the Debian security tracker and indeed all the security fixes were already applied to our Raspbian Stretch 9.11.

7.2 Snort crash - registered rules

As explained in 5.3.3, we had to drop the first rule set (i.e., registered rules - Figure 5.5) because of a crash happening during the initialization process of Snort. Indeed, the operating system was killing Snort process every time we launched it. We used the *top* command which gives us an overview of the different process running (see Figure 7.1).

We observe the CPU utilization, for the Snort process, to be extremely high (i.e., 99%) and in the following seconds the process would be killed by the operating system. It seemed odd to us because It was more likely to have memory issues, since the rule set is big and the KUNBUS Revolution Pi connect+ has only 1GB of RAM. Therefore we used the *dmesg* command which "print or control the kernel ring buffer"[53].

The output of *dmesg*, figure 7.2, helps us to understand the real reason of Snort killing. Indeed, all modern Linux kernels have a built-in mechanism called *Out-Of-Memory* which can annihilate your processes under extremely low memory conditions. When such a condition is detected, the killer is activated and picks a process to kill. The target is picked using a set of heuristics scoring all processes and selecting the one with the worst score to kill. In this case, the score for the Snort process is 451 which is the highest score.
Hence, the *registered rules* rule set is too memory demanding for our device because each rule has to be parsed and pre-processed before starting packet analysis.

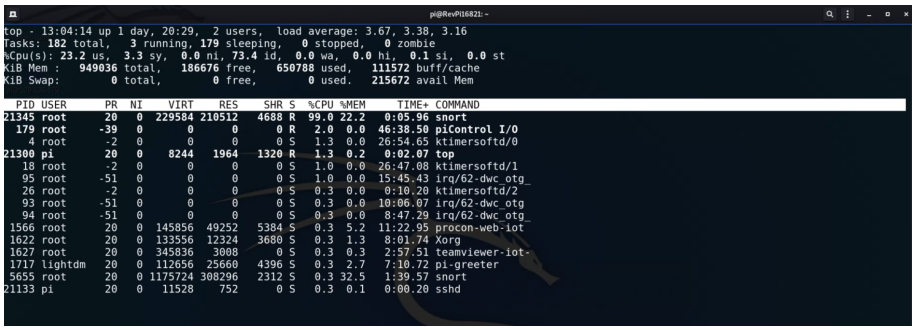

Figure 7.1: top command with Snort process CPU and memory usage

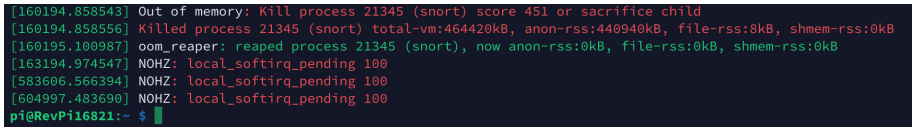

Figure 7.2: dmesg output

7.3 Network traffic analysis

In section 6, we displayed the results of the *nload* network traffic tool. The network traffic before the attack is normal with only the default services running on the

Revolution Pi connect+.

During the attack, the network traffic observed is unusually high for the outgoing traffic. Indeed, normally the attack should be observed on the incoming traffic. However, due to technical limitations, we were forced to use *tcpreplay* in order to replay the attack directly on the host (i.e., Revolution Pi connect+). Therefore, *nload* detected the traffic as going out as it was sort of generated by the local machine.

In order to verify if the attack was indeed reaching our local host (i.e., 192.168.1.54) on port 443, we used *tcpdump* on the Revolution Pi connect+. It allowed us to check that the Revolution Pi connect+ (i.e., 192.168.1.54) was indeed being targeted by the Kali Linux machine (i.e., 192.168.1.41) with the Loïc attack we captured on a *pcap* file.

On Figure 7.3, we can notice two things: first we observe that the Revolution Pi connect+ (i.e., 192.168.1.54) is being busy by TCP connections on port 443 sent by the Kali Linux machine (i.e., 192.168.1.41). Second, at the end of the *tcpdump* output, the number of *packets captured*, *packets received by filter* and *packets dropped by kernel* is displayed. The number of *packets dropped by kernel* is very high (i.e., 114718) which can be explained and correlated by the CPU usage. Indeed, the CPU is overloaded and the process in charge of handling the network connection cannot handle the amount of connections coming in on port 443.

7.4 Snort limitations analysis

The results we obtained in chapter 6 concerning Snort limitations will be analyzed in this section.
It is important to remember that because of memory limitations, explained in section 7.2, the Revolution Pi connect+ is running a basic rule set (i.e., community rules). All rules are set to **Alert** only. Therefore, Snort should not be dropping packets.

In order to investigate more about those packets dropping, we reduced the number of rules on the *community.rules* file. Instead of having approximately 10.000 rules, we kept only 10 rules. The idea was to compare the number of packets dropped by Snort using the 10.000 rules set and the 10 rules set.
In fact, the number of packets dropped by Snort using the 10 rules set is much lower (see Figure 7.4) than the *141519* dropped for the second rule set (i.e., 10.000 rules). Indeed, just *5528* packets were dropped.

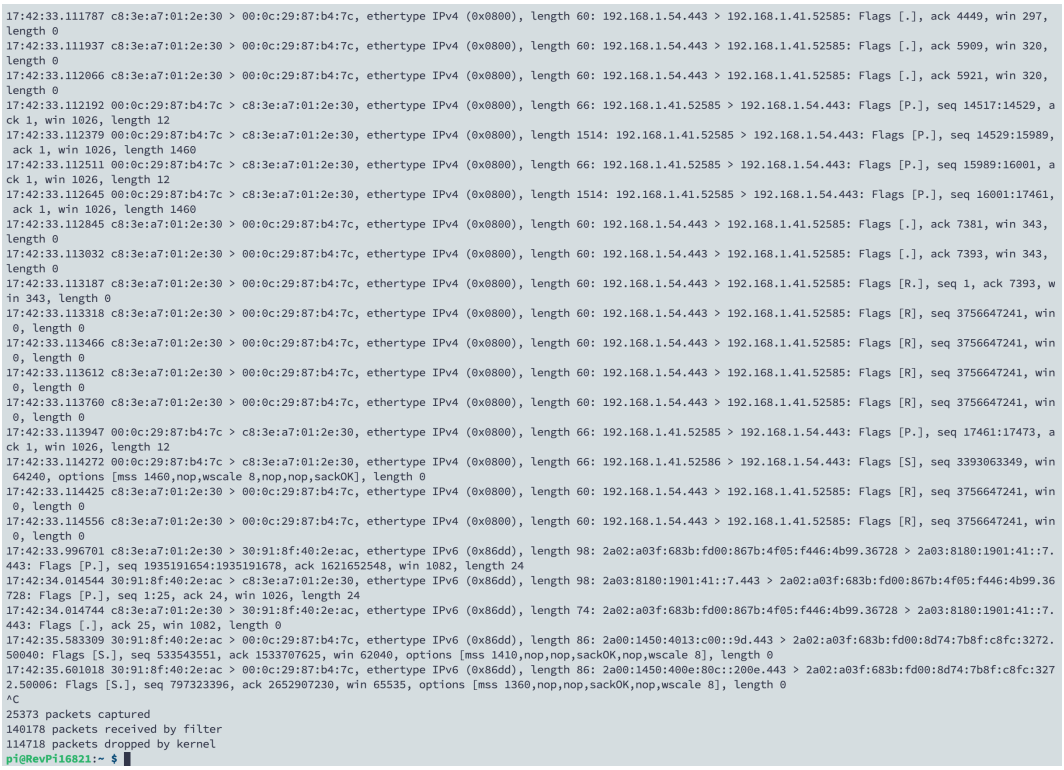

Figure 7.3: tcpdump output

Nevertheless, the number of dropped packets is still significant. In a real world scenario, the number of rules can be substantially higher in order to be protected against the most known and recent threats.

Therefore, Snort limitations on the Revolution Pi connect+ seem obvious and are critical in order to run an IDS on the Revolution Pi connect+.

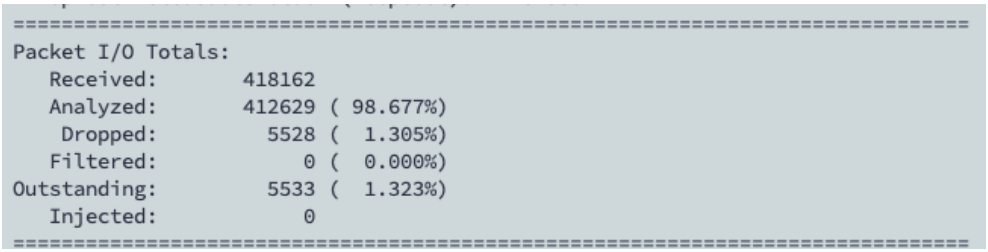

Figure 7.4: Number of packets dropped using the 10 rules set

7.5 MQTT delays analysis

From the delays encounter in section 6.5, further investigation was made to understand the device limits.
This shows that in the network side, the Revolution Pi connect+ has some troubles processing data continuously without encountering delay while a lot of incoming requests are being received. The problems start around 10 megabytes per second and needs to be taken into consideration for implementation of systems or software which needs to receive information periodically. The control of the bandwidth could play an interesting role in avoiding this issue.

7.6 LED switching

In the contrary of section 7.5, the physical connections are very reliable. In fact, both devices were working well and doing their job. This can be explained by the importance KUNBUS company has given into the hardware implementation to ensure strong and dependable modules.
Furthermore, the program handling the communication between the Revolution Pi connect+ and DIO named *pl011_pio_tx* has a high default priority (PR) of -51 which cannot be easily changed.

7.7 Disk image cloning analysis

As described in the previous Chapter, where the results of the disk imaging were displayed, we were astonished by the ease of such an operation. Indeed, no password or credential was asked for the cloning process and all the files that were stored on the Revolution Pi connect+ hard disk were mounted and displayed in clear-text.
Consequently, the confidentiality of data stored in such a device is not always honored. There is a risk of a physical data breach attack, where a malicious person with physical access to the connect+ could copy all data into another device. This means that such a device needs to be placed in restrictive areas where only authorized people can be in contact with it.
Additionally, as we explained in Chapter 5 Section 5.7, the connect+ is shipped with a crypto-chip that could be used to encrypt the entire hard disk but it is not the case.

Chapter 8

Conclusions

To conclude, development wise the Revolution Pi modules offer an unlimited number of possibilities with great potential. It gives the opportunity to create a small module-based network with the Revolution Pi core or connect+ as the heart of it. This network can be fully customised and used for multiple different goals. Nonetheless, the security aspect stays essential. Indeed, as much as it is possible to develop great projects using these devices, there is also the possibility of having important or critical flaws in the system. The company insists on the possibility of using the module as gateways which increases the risks of infection and spreading if one machine is infected or attacked.

In that matter, even as a gateway the device still has some limits. In fact, difficulties have been observed on the machine behaviour during a DoS attack and both physical and wireless flaws have been discovered through the pre-opened services of the device. In addition, even if the organisation state their project as being an open-source one, a few secrets were kept as for example the crypto chip *ATECC508A*. Deeper investigations could be made since it is possible that other secrets hidden from the community resides in the modules.

On the other hand, the machines robustness is impressive. In fact, surprisingly there was no problems with any other physically linked modules when they were put under pressure. The Revolution Pi DIO continued to work well with the LED switching on and off without any delay and responded directly to any user interaction. Indeed, even with the limited resources it has compared to a real server or computer, its reaction and stability were impressive. The problem resided more into the network side and the constant exchange of packets.

There is still a lot to find out about KUNBUS modules security wise. In fact, not

only KUNBUS Cloud could not be fully analysis but also as in this paper only two devices were assembled and analyse, a fully complex Revolution Pi network with specific behaviour or purpose could not be tested. Furthermore, the organisation seems to hide a few things and approaching them to grow better understanding in the modules could be useful. There will always be space for security improvement and it applies even more when the only possible limit of a device is the human imagination.

Appendix A

Appendix

A.1 Publisher script

```
1  import random
2  import time
3  import paho.mqtt.client as mqtt
4
5  def on_publish(client,userdata,result):
                #create function for callback
6      print("data published \n")
7
8  client1= mqtt.Client("publisher1")
                                #create client
       object
9  client1.on_publish = on_publish
10 client1.username_pw_set("ED", password=None)
                               #assign function
       to callback
11 client1.connect("192.168.1.26",1883)
                                        #establish
       connection
12 client1.loop_start()
13
14 while True:
15
16     msg_to_be_sent = random.randint(0, 100)
17     client1.publish("test/sensor",
18                     payload=msg_to_be_sent,
19                     qos=0,
```

```
20                              retain=False)
21
22              time.sleep(0.5)
23          client1.loop_stop()                     #publish
```

A.2 Switch LED on/off script

```
1           import Revolution Pimodio2
2           import time
3
4           rpi = Revolution Pimodio2.Revolution PiModIO(
                autorefresh=True)
5
6           while True:
7               if(rpi.io.O_2.value == False):
8                       rpi.io.O_2.value = True
9               else:
10                      rpi.io.O_2.value = False
11              time.sleep(0.5)
```

List of Figures

Bibliography

[1] Javier Lopez Rodrigo Roman, Pablo Najera. Securing the internet of things. *IEEE Computer*, 44:51–58, 2011.

[2] Tomas Gea, Josep Paradells, Mariano Lamarca, and David Roldan. Smart cities as an application of internet of things: Experiences and lessons learnt in barcelona. In *Proceedings of the 2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 552–557, USA, 2013. IEEE Computer Society.

[3] Dr. Yusuf Perwej, Kashiful Haq, Dr. Firoj Parwej, and Mumdouh M. The internet of things (iot) and its application domains. *International Journal of Computer Applications*, 182:36–49, 04 2019.

[4] Amy Nordrum. Ieee spectrum,Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated, 2016. `https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated`.

[5] Kayla Matthews. Iot for all, The Evolution of IoT Hacking, 2018. `https://www.iotforall.com/evolution-iot-hacking/`.

[6] INFSO D.4 Networked Enterprise, RFID INFSO G.2 Micro, and Nanosystems. Internet of things in 2020, roadmap for the future. In *Co-operation with the Working Group RFID of the ETP EPOSS*, 2008.

[7] Chawngsangpuii R, Das Prodipto, and Kumar Das Rohit. Security management perspective for internet of things. *International Journal of Engineering Science Invention*, 6:59–65, 09 2017.

[8] i SCOOP. Iot gateways and industrial iot gateways – usage and evolutions, 2017. `https://www.i-scoop.eu/internet-of-things-guide/iot-gateways/`.

[9] Nuno Souto Andre Gloria, Francisco Cercas. Design and implementation of an iot gateway to create smart environments. *Procedia Computer Science*, 12 2017.

[10] Trend Micro. Industrial internet of things (iiot). `https://www.trendmicro.com/vinfo/us/security/definition/industrial-internet-of-things-iiot`.

[11] Saurabh Singh, Pradip Kumar Sharma, Seo Yeon Moon, and Jong Hyuk Park. Advanced lightweight encryption algorithms for iot devices: survey, challenges and solutions. *Journal of Ambient Intelligence and Humanized Computing*, pages 59–65, 05 2017.

[12] Meltem Sönmez Turan Nicky Mouha Kerry A. McKay, Larry Bassham. Report on lightweight cryptography. Technical Report 8114, National Institute of Standards and Technology, 2017.

[13] Juan Ruiz Lagunas, Antolino-Hernandez Anastacio, Reyes-Gutierrez Mauricio, Ferreira-Medina Heberto, Torres-Millarez Cristhian, and Paniagua-Villagomez Omar. How to improve the iot security implementing ids/ips tool using raspberry pi 3b+. *International Journal of Advanced Computer Science and Applications*, 10, 01 2019.

[14] Ranjith. Autosploit : Automated mass exploiter, 2019.

[15] Sanaz Rahimi Moosavi, Tuan Nguyen gia, Amir M. Rahmani, Ethiopia Nigussie, Seppo Virtanen, Jouni Isoaho, and Hannu Tenhunen. Sea: A secure and efficient authentication and authorization architecture for iot-based healthcare using smart gateways. *Procedia Computer Science*, 52, 12 2015.

[16] Mališa Vučini, Bernard Tourancheau, Franck Rousseau, Andrzej Duda, Laurent Damon, and Roberto Guizzetti. Oscar: Object security architecture for the internet of things. *Ad Hoc Netw.*, 32:3–16, 09 2015.

[17] S. Raza, S. Duquennoy, T. Chung, D. Yazar, T. Voigt, and U. Roedig. Securing communication in 6lowpan with compressed ipsec. In *2011 International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS)*, pages 1–8, June 2011.

[18] Alessandro Sforzin, Félix Gómez Mármol, Mauro Conti, and Jens-Matthias Bohli. Rpids: Raspberry pi ids — a fruitful intrusion detection system for iot. *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications,*

Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), pages 440–448, 2016.

[19] Andreas Aspernas and Thommy Simonsson. Ids on raspberry pi, a performance evaluation. Master's thesis, Linnaeus University, 2015.

[20] Weizhe Zhang, Bin Zhang, Ying Zhou, Hui He, and Zeyu Ding. An iot honeynet based on multiport honeypots for capturing iot attacks. *IEEE Internet of Things Journal*, PP:1–1, 11 2019.

[21] Suha Ibrahim Al-Sharekh and Khalil H. A. Al-Shqeerat. Security challenges and limitations in iot environments. In *JCSNS International Journal of Computer Science and Network Security*, volume 19, pages 193–199, February 2019.

[22] PLCopen. Iec 61131-2: Equipment requirements and tests, 2017. `https://plcopen.org/iec-61131-2`.

[23] Andreas Schmidt. *Secrecy versus openess: Internet security and the limits of open source and peer production*. Uitgeverij BOXPress, 60594 Frankfurt am Main, Germany, 2014.

[24] Tom Warren. Microsoft: we were wrong about open source. `https://www.theverge.com/2020/5/18/21262103/microsoft-open-source-linux-history-wrong-statement`.

[25] Revolution Pi. Digital io modules, 2018.

[26] Kunbus entreprise. Pibridge – how communication between revolution pi modules works. `https://revolution.kunbus.com/pibridge/`.

[27] Kunbus entreprise. Revpimodio - revolutionpi with python3! `https://revpimodio.org/en/homepage/`.

[28] Cisco. Cisco completes acquisition of sourcefire, Acquisitions, 2013. `https://www.cisco.com/c/en/us/about/corporate-strategy-office/acquisitions/sourcefire.html`.

[29] William Stallings. *Network Security Essentials: Applications and Standards (4th Edition)*. Prentice-Hall, Inc., USA, 2011.

[30] Paul E. Proctor. *The Practical Intrusion Detection Handbook*. pub-PHPTR, pub-PHPTR:adr, 2000.

[31] Wikipedia. Système de détection d'intrusion. https://commons.wikimedia.org/wiki/File:Topologie_NIDS_HIDS.png?uselang=fr.

[32] Dan Dinculeana. Vulnerabilities and limitations of mqtt protocol used between iot devices. *Applied Sciences*, 9:848, 02 2019.

[33] BehrTech. What is mqtt and why you need it in your iot architecture, 2019. https://behrtech.com/blog/mqtt-in-the-iot-architecture/.

[34] Oasis Standard. Mqtt version 5.0, 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901002.

[35] Oasis Standard. Mqtt version 5.0, 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html#_Toc3901261.

[36] Ed. T. Ylonen, C. Lonvick. The secure shell (ssh) protocol architecture, 2006. https://www.hjp.at/doc/rfc/rfc4251.html.

[37] J. Mogul H. Frystyk L. Masinter P. Leach T. Berners-Lee R. Fielding, J. Gettys. Hypertext transfer protocol – http/1.1, 1999. https://www.hjp.at/doc/rfc/rfc2616.html.

[38] E. Rescorla T. Dierks. The transport layer security (tls) protocol version 1.2, 2008. https://www.hjp.at/doc/rfc/rfc5246.html.

[39] Forrester. The forrester wave™: Vulnerability risk management, q4 2019, 2019. http://resources.mynewsdesk.com/image/upload/vxhezwigxqgzboeviblh.pdf.

[40] Rapid7. How to install snort nids on ubuntu linux, 2017. https://blog.rapid7.com/2017/01/11/how-to-install-snort-nids-on-ubuntu-linux/.

[41] Cisco. Snort - network intrusion detection system, 2020. https://www.snort.org/.

[42] Gaurang Pandya. Preparing to withstand a ddos attack, November 2015. https://www.sans.org/reading-room/whitepapers/incident/preparing-withstand-ddos-attack-36412.

[43] Subramani Rao. Denial of service attacks and mitigation techniques: Real time implementation with detailed analysis, September 2011. https://www.sans.org/reading-room/whitepapers/detection/denial-service-attacks-mitigation-techniques-real-time-implementation-detailed-analysis-33764.

[44] Aaron Turner. Tcpreplay official website. http://tcpreplay.appneta.com/.

[45] Kunbus. Revolution pi forum. https://revolution.kunbus.de/forum/.

[46] John. Revolution pi forum: security concerns. https://revolution.kunbus.de/forum/viewtopic.php?f=6&t=906.

[47] Microchip. Atecc508a. https://www.microchip.com/wwwproducts/en/ATECC508A#additional-features.

[48] Kunbus entreprise. Crypto chip. https://revolution.kunbus.com/forum/viewtopic.php?t=792.

[49] NIST. National vulnerability database, 2020. https://nvd.nist.gov/vuln/search.

[50] Linux Die. nload(1) - linux man page, 2012. https://linux.die.net/man/1/nload.

[51] marcelm. How do i update apache2 to the latest version on debian jessie?, 2019. https://unix.stackexchange.com/questions/404036/how-do-i-update-apache2-to-the-latest-version-on-debian-jessie.

[52] Debian. Security bug tracker, 2019. https://security-tracker.debian.org/tracker/.

[53] Linux.die. dmesg(8) - linux man page, 2020. https://linux.die.net/man/8/dmesg.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**