UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Internet of Things security and privacy

Exploration of vulnerabilities and extension of an existing honeypot

Dissertation presented by
Quentin GROULARD, Yvhan SMAL

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Ramin SADRE

Reader(s)
Lionel METONGNON, Pierre REINBOLD

Academic year 2017-2018

Abstract

Internet of Things (IoT) is the new trend in this connected world. The number of devices used for this purpose is exploding since several years. In addition, new protocols, hardware designs and Operating Systems emerged and are still emerging to respond effectively to the needs of this new trend. These innovations, as a lot of new vulnerabilities make their appearance, turn out to be a true opportunity for attackers to put their knowledge into practice and allowing them to do what they do best: obtain, destroy, or even reveal information without authorized access or permission.

In a first step, this thesis aims at presenting common IoT technologies by making a survey of them and exposing some of the vulnerabilities they present grouped into well-known attacks. In a second step, the goal is to extend a low interaction honeypot based on Message Queuing Telemetry Transport (MQTT) by improving its level of interaction, trying to identify new kinds of attacks on the protocol and exposing the way of proceeding from an attacker viewpoint.

The results obtained from the interactions with the honeypot are not in great numbers. However, they allow to understand the basic process conducted by attackers targeting MQTT. As the excitement for IoT is predicted to grow even more in the future, the one for MQTT should also follow the trend. For this reason, the honeypot might come in very handy in the process of fooling attackers that want to interact with the protocol and gathering interesting knowledge about their actions.

Acknowledgements

We would like to express our gratitude to our supervisor Professor Ramin Sadre for its availability and the advice he offered us during this thesis. He put us on track and was always of great help when we were struggling.

We also would like to thank Lionel Metongnon for the logical support and his comments in addition to be one of the readers of this thesis. As well as Pierre Reinbold, the second reader.

Quentin I would personally like to thank my family for their unconditional support and especially my parents for the outstanding education they gave me. Lindsay, for her listening ear and the moments of happiness we share. And all the people who crossed my path during my time at the University and contributed to make it unforgettable: friends, flatmates, classmates and last but not least, the XXVII Board of BEST Louvain-la-Neuve.

Yvhan I would like to especially thank my family and my girlfriend who were always there to cheer me up during difficult times these last five years. Also, thank you to all the people that I've crossed paths with and who have made my university journey an incredible experience.

Contents

1 Introduction **3**
- 1.1 Context . . . 3
  - 1.1.1 What is IoT? . . . 3
  - 1.1.2 Key characteristics . . . 4
  - 1.1.3 Challenges . . . 4
- 1.2 Objectives . . . 5
- 1.3 Outline . . . 5

2 IoT technologies **6**
- 2.1 Architecture of IoT . . . 6
  - 2.1.1 Perception layer . . . 6
  - 2.1.2 Network/Transportation layer . . . 7
  - 2.1.3 Application layer . . . 7
- 2.2 Common IoT technologies . . . 7
  - 2.2.1 Infrastructure . . . 7
  - 2.2.2 Communication/Transport . . . 9
  - 2.2.3 Multi-layer . . . 10
  - 2.2.4 M2M communication . . . 11
- 2.3 Vulnerabilities in IoT technologies . . . 13
  - 2.3.1 Denial-of-Service . . . 13
  - 2.3.2 Cryptographic key compromising . . . 14
  - 2.3.3 Violation of privacy . . . 15
  - 2.3.4 Impersonation . . . 17
  - 2.3.5 Botnets . . . 19

3 Extension requirements **23**
- 3.1 Deeper look into MQTT . . . 23
  - 3.1.1 Quality of Service . . . 24
  - 3.1.2 Connection establishment . . . 25
  - 3.1.3 Subscribe, Unsubscribe & Publish . . . 26
  - 3.1.4 MQTT v5 . . . 28
- 3.2 Honeypots . . . 30
  - 3.2.1 What's a honeypot? . . . 30
  - 3.2.2 Examples of honeypots for IoT . . . 31
  - 3.2.3 Dionaea . . . 33

4 Dionaea honeypot extension **35**
4.1 Previous MQTT module . . . 35
4.2 New MQTT module . . . 35
4.2.1 Packets . . . 35
4.2.2 Connection establishment . . . 36
4.2.3 Subscription to a topic . . . 37
4.2.4 Unsubscription from a topic . . . 38
4.2.5 Publication to a topic . . . 38
4.2.6 Quality of Service . . . 38
4.2.7 Disconnection . . . 39
4.2.8 Broker information topic . . . 39
4.2.9 Logging to the database . . . 39
4.2.10 Summary . . . 40

5 Evaluation **41**
5.1 Tools . . . 41
5.1.1 Oracle VM VirtualBox . . . 41
5.1.2 Wireshark . . . 41
5.1.3 Eclipse Paho . . . 41
5.1.4 mqtt-spy . . . 42
5.1.5 MQTTBox . . . 42
5.2 Testing . . . 42
5.2.1 Conformity tests . . . 42
5.2.2 Load tests . . . 44
5.3 Results . . . 45
5.3.1 MQTT related results . . . 45
5.3.2 General results . . . 46
5.4 Attack simulation . . . 48
5.4.1 Find an open MQTT broker . . . 49
5.4.2 Establish a connection using a MQTT client . . . 49
5.4.3 Obtain broker related information . . . 49
5.4.4 Observe exchanged messages through the broker . . . 50

6 Recommendations **52**

7 Conclusion and future work **54**

A Install and run the honeypot **56**

B Run the tests **57**

Chapter 1

Introduction

1.1 Context

1.1.1 What is IoT?

Everybody knows what the Internet is as of today but for the IoT, that's a different kettle of fish. When searching on the web for a definition of IoT, people can find themselves confused due to their diversity. Basically, the IoT can be seen as an extension of the Internet that we all know. The *things* represent ordinary objects of our everyday life which are connected to the Internet and that exchange information with others of its own kind. And to accomplish a specific goal, they make decisions based on information retrieved by embedded sensors and actuators allowing to capture information from the real world and to interact with it.

Which goals? Well, IoT turns out to be useful for a lot of applications allowing us to be more efficient in the way we accomplish specific tasks but also to save us time, money and to increase our productivity. Here are some sectors where IoT is used and related examples:

- **Home automation**: Automatic control of home equipment such as an intelligent door equipped with a camera able to detect the house owners at the frond door and automatically unlocking itself if the person is recognized.
- **Healthcare**: Doctors remotely monitor patients who use wearables able to give information about their health condition preventing them from going to the doctor to do a diagnosis.
- **Agriculture**: Farmers monitor their crop fields thanks to devices with sensors that sense the light, humidity, soil moisture, temperature, etc. that can trigger an automatic irrigation system.
- **Supply chain management**: Goods that are tracked while they are on the move from the vendor to a manufacturing facility allowing the latter to know if there is any delay and thereby adapting its production accordingly.

Is it important to care about this new paradigm? According to a collection of IoT forecasts given by Forbes [11] in late 2017, the number of IoT devices is estimated to grow to almost 31 billion worldwide by 2020 and the global IoT market to grow to 661.74 billion by 2021. Moreover, no less than 65% of enterprises are said to take into account IoT products within their strategy by 2020. Following the new rule for the future that "Anything that can be connected, will be connected", it seems very important to stay abreast of this trend.

1.1.2 Key characteristics

IoT can only be realized by the deployment of multiple technologies. The following key characteristics grouped together are the ones that help to bridge the gap between the virtual and physical world [4, 37, 39, 44].

- **Identification**: The function of identification is to map a digital identity to a particular object or thing such that when it produces information, we know for sure where the data is coming from. Moreover, we may also want to retrieve a set of features from a particular thing in some cases. Identification turns out to be very useful for the design and architecture of IoT networks but also for managing security operations.
- **Low power and energy**: Devices at the very bottom of the IoT pyramid are constrained objects limited in terms of energy, storage and computing. They thus need to manage their resources in the most efficient way possible in order to perform their task of sensing the environment and monitoring incoming events.
- **Interconnectivity**: As they are limited regarding resources, devices have the requirement to be interconnected with the global information and communication infrastructure in order to convey information at the right place. Concerning things that are more advanced and less constrained, these need to be interconnected by the means of the Internet, allowing to capture data from others to accomplish the task they were designed for.
- **Dynamic changes**: IoT has to be open by definition, allowing automatic adjustments. The number of devices often changes as some are dynamically joining and leaving the network, others sleep for a specific period of time to save some resources and then wake up when information gets to them.
- **Huge scale**: As said before, the number of IoT devices is estimated to grow to almost 31 billion worldwide by 2020. The amount of data produced by all these devices needs to be interpreted for application purposes which means that it must be efficiently handled and processed before.

1.1.3 Challenges

The challenges facing the IoT are numerous, the two most-talked about being probably the *security* and *privacy* ones. Even if more and more people are made aware, there is still a lot of improvement to be made on these sides.

Security

Security can be presented as a significant challenge mainly because of the interconnectivity between things. This interconnectivity is characterized by a big and, most of the time, wireless Ad-Hoc network. An attacker could enter this network by spotting and exploiting a vulnerability in a specific connected device then having an easier time to compromise other devices. Poorly secured IoT devices can serve as an entry point for unprecedented cyber-attacks.

Another important security challenge for IoT concerns the combination of different existing technologies. In fact, IoT can be broken down into several layers. Each of these layers has a task and a set of available technologies to fulfill the task. The chosen technologies must be full-blown secured as any security vulnerability they present could be propagated to technologies used in other layers. However, it is not always the case and as the majority of these technologies relies on an open trust model, only one small vulnerability is enough to break and entire IoT system.

The big majority of IoT is composed of resource-constrained devices having limited battery, computation power and storage capabilities. Thus, they are often unable to implement good security mechanisms as the entire resources in their possession are used to complete their primary tasks of sensing, actuating and communicating with other devices. Manufacturers of IoT devices tend to keep their design as simple as possible by avoiding to add features that are not necessary for their major purpose.

Privacy

Users are often driven to enter some personal information (e.g. location, health condition or habits) by connected devices or applications for producing a huge load of data afterwards and often, these information are not well maintained away from people that are not supposed to have access to them. It is also the case that people are mislead or even not fully aware of what is collected through their smart things and how it is used as IoT can be very ubiquitous.

1.2 Objectives

This thesis aims at fulfilling two objectives that follow a logical development, the first one being a necessary step for the second one.

The first objective is to study the possible points of attack in commonly used IoT technologies and define a list of IoT attacks that can occur according to the mentioned points of attack.

The second objective consists in clearly understanding how IoT attacks are performed in practice and how to defend against them thanks to a honeypot. Moreover, the IoT honeypot used is intended to be an extension of an existing honeypot such that it emulates an IoT-specific technology.

1.3 Outline

Chapter 2 lays the foundations of IoT by first explaining a high-level kind of architecture for the entire IoT. Commonly used technologies in this field are then explained and grouped according to the problem(s) they are supposed to solve. Following is a discussion about some vulnerabilities and security issues that previously mentioned technologies present.

Chapter 3 takes a deeper look at the MQTT functionalities as it is the main technology that allows the honeypot to be extended. The following section gives an overview of honeypots and explain in greater detail the dionaea honeypot used as a material.

Chapter 4 makes the comparison between the MQTT module already present in dionaea and the one that makes up our extension by giving an explanation on how every functionalities were added or improved.

Chapter 5 first describes the tools we used in the honeypot extension process. Second section exposes the way we tested the implementation of the MQTT module extension to make sure it is reliable. Then, MQTT related and general results obtained thanks to the interaction of other people with the honeypot are discussed before demonstrating, with an attack scenario, how one of the basic attacks can be conducted against the MQTT protocol.

Finally, chapter 6 gives few recommendations for manufacturers, developers as well as consumers in order to avoid the potential harm that can be done using IoT technologies.

Chapter 2

IoT technologies

This chapter gives an overview of what IoT is constituted of, from standards used in the infrastructure of IoT networks to communication protocols used in these networks.

The first section presents a high-level kind of architecture for the entire IoT. Commonly used IoT technologies are explained in the second section before discussing about some vulnerabilities and security issues they present in the third section. Concerning the considered technologies, we limit ourselves to the ones for which we found significant vulnerabilities.

2.1 Architecture of IoT

As there is no standard stack concerning IoT, we can see different ones being suggested having from three to five layers. Here, we will mainly focus on an architecture having three layers as presented by Jing et al. [29] and pictured in Figure 2.1.

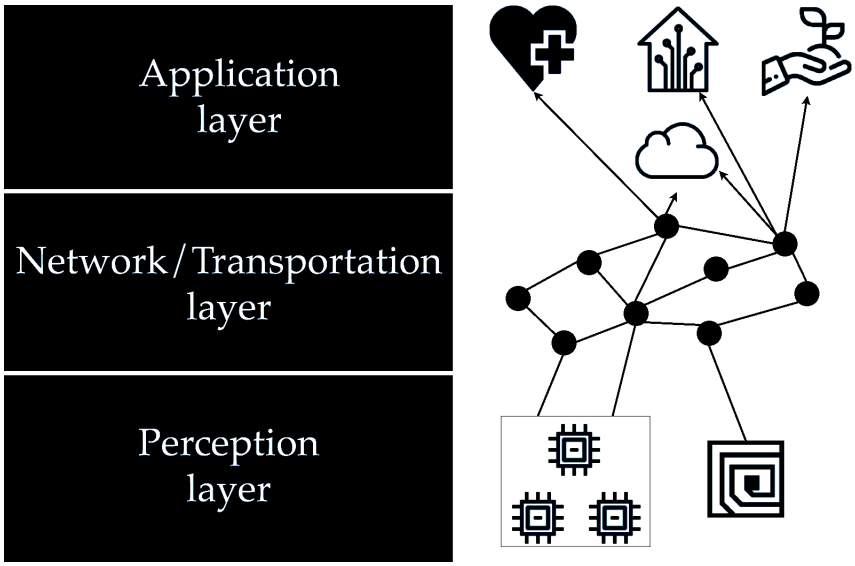

Figure 2.1: IoT 3-layers architecture

2.1.1 Perception layer

It consists of the physical objects called *sensors* or *devices* equipped with low computation power and storage capacity. These interact with the physical world by collecting specific information like location, temperature, pressure, humidity, vibration etc.

This layer also includes the perception network whose task is to bring the information from sensors nodes to a gateway capable of sending it through the next layer, which is the *network/transportation layer*.

As the sensors produce a huge load of information, gateways sometimes perform a task of data pre-processing (e.g. data aggregation) in order to save storage and computation resources required at the upper layer. This practice is known as fog computing [6].

2.1.2 Network/Transportation layer

Network/transportation layer basically transfers data produced by the *perception layer* to the layer right above, the *application layer*. To do so, multiple communications technologies for wireless/wired networks are coupled together in a heterogeneous way.

2.1.3 Application layer

This last layer processes data that arrives from the layer underneath before realizing intelligent decisions (thanks to cloud computing in particular) like knowing where the data must go, determining whether it is malicious etc. It is then responsible for delivering data to various supported applications (e.g. smart health care, smart home, smart agriculture) also included in this layer. *Application layer* is mainly seen as the front end of the whole IoT architecture through which IoT possibilities are exploited.

2.2 Common IoT technologies

Commonly used technologies in this field are explained in this section and grouped according to the problem they are supposed to solve.

2.2.1 Infrastructure

Infrastructure refers to the way nodes in IoT networks connect themselves together as well as how the information produced is supposed to be represented and to flow in these networks.

IEEE 802.15.4

IEEE 802.15.4 [21] is the standard for Low-Rate Wireless Personal Area Network (LR-WPAN). Mainly used in residential and industrial environments applications, its main features consist of network flexibility, low cost, and low power consumption. It defines the physical and Medium Access Control (MAC) layers for the concerned networks, leaving the freedom to other protocols/standards (e.g. IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN) or ZigBee which are introduced later) to define the upper layers.

It supports two types of nodes no matter which network topology (star, mesh or cluster tree): Fully Functional Devices (FFDs) which have full network functionalities and that are able to act as Personal Area Network (PAN) coordinators and the Reduced Functional Devices (RFDs) which have limited functionalities. In every PAN there is at least one FFD. The task of RFDs is to get the data from the physical world and transmit them to a PAN coordinator.

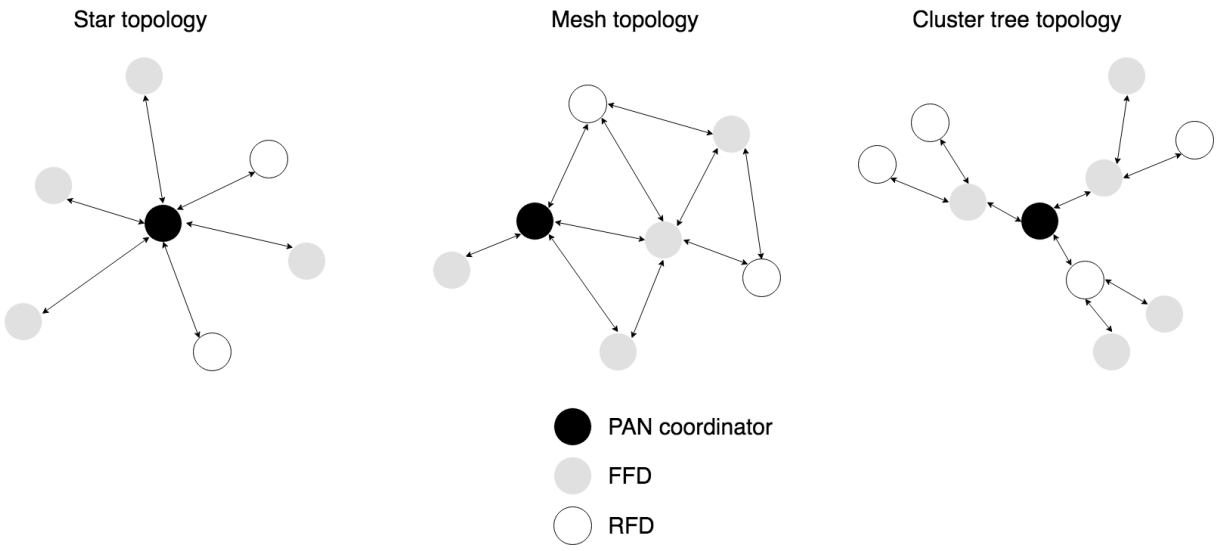

Figure 2.2: Network topologies for 802.15.4

6LoWPAN

6LoWPAN [56] refers to a protocol stack enabling 802.15.4 nodes to communicate with each other using Internet Protocol version 6 (IPv6) technology. In fact, using IPv6 for the IoT would be much easier for everyone as this is a well-known technology having a large address space and supporting multiple kinds of link technologies. It makes the use of IPv6 possible for IoT by providing an adaptation layer on top of 802.15.4 that performs header compression, packet fragmentation into frames and link layer forwarding.

RPL

Considered as the default routing option for IPv6 resources constrained networks, Routing Protocol for Low-Power and Lossy Networks (RPL) [65] allows to efficiently route data in a distance vector way in IoT sensors networks. Its job is to establish a network topology on top of Low-power and Lossy Networks (LLNs) by building multiple Destination Oriented Directed Acyclic Graphs (DODAGs), each having at their root a server, border router, gateway or sink. In this DODAG, all the links are oriented in such a way that no cycle exist. A RPL instance is characterized by multiple disjoint DODAGs whose union is a Directed Acyclic Graph (DAG). Each RPL node can be part of at most one DODAG in a RPL instance, and this is the reason that makes every DODAG disjoint.

Multiple RPL instances can run independently within a single network by each having a different objective function. This latter is used to compute the rank value of a node which defines the node's position to the DODAG root based on a specific metric. Rank strictly decreases in the up direction towards the root and strictly decreases in the down direction towards the leaves. There are three types of messages used in RPL:

- **DODAG Information Object (DIO)**: travels downwards periodically or on request with the rank of sending node for nodes below to find a parent;
- **Destination Advertisement Object (DAO)**: travels upwards to the root to propagate information about children such that parents can populate their routing tables;
- **DODAG Information Solicitation (DIS)**: used when a new node wants to join an RPL instance, when a node wakes up and requires information about the network/informs that it is back or when it is looking for a new parent.

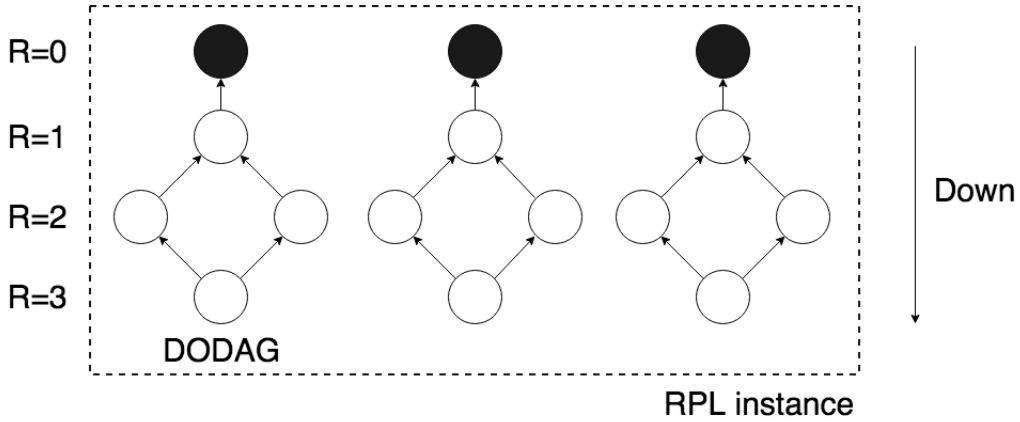

Figure 2.3: RPL ranks, DODAG, instance with child-parent relationship indicated by arrows.

The root has the capability of advertising a new DODAG thanks to a DIO message that includes a version number. A node joining a DODAG with version X will ignore versions smaller than X and when a node receives a message with a higher version, it is able to join the new DODAG by getting a new parent along with a new rank.

RPL also provides a way to avoid sending "unnecessary" DIO messages with the trickle timer [32]. Its task is to allow nodes exchanging messages in a simple and scalable way by dynamically adjusting the transmission window. Only a few messages are sent when the network is considered stable.

2.2.2 Communication/Transport

Technologies described next are communication technologies used to move the information produced from one point to another.

RFID

The basic use for Radio-Frequency Identification (RFID) [63] is to identify items with tags. It is a very old technology as it exist since the forties. An RFID tag is a small device that contains a microcircuit, an antenna and an identification number. Attached to or incorporated into a product, an animal or a person, it is able to emit messages requested by an RFID reader through wireless data transmission. It can have a writable memory to store information that can later be communicate to a reader. Frequency ranges go from a few centimeters up to 100 meters.

There are three categories of tags: active, passive and semi-passive tags. Active tags have their own source of power obtained thanks to a battery whereas passive tags obtain energy from a reader's electromagnetic field. Semi-passive tags are between the two as they have a battery for internal calculations but they need energy from a reader's electromagnetic field to transmit information.

NFC

Near Field Communication (NFC) [14] is a subset of the RFID technology. It is a more sophisticated technology as NFC devices can act both as a tag and as a reader and they can exchange data in a secure form. They operate at the same frequency than High-Frequency RFID devices (13.56 MHz) but must be in close range from each other to communicate. NFC is extensively used by credit or debit cards for contactless payment for example.

Bluetooth Low Energy

Bluetooth Low Energy (BLE) [59] is wireless technology for short-range communication as an integral part of Bluetooth 4.0 [59]. As the name implies, it's a version of Bluetooth focusing on low energy consumption for controlling and monitoring applications rather than focusing on high data rate as in classical Bluetooth. This is possible by sending small chunks of data with a small overhead when it's required and putting the connection in sleeping mode when no data needs to be exchanged.

Bluetooth 4.0 includes two new categories of Bluetooth devices: Bluetooth Smart and Bluetooth Smart Ready. The first one indicates a single mode BLE requiring either a Smart Ready or another Smart device to communicate while the second one indicates a dual mode BLE compatible with classic and Bluetooth Smart devices.

2.2.3 Multi-layer

Multi-layer technologies implement solutions for multiple layers in the IoT architecture.

Zigbee

Based on the 802.15.4 standard as 6LoWPAN, Zigbee [42] is an open global standard developed by the Zigbee Alliance[1] for wireless mesh networks that are low-power, low-cost and low-data-rate. It allows sensors monitoring and control of various applications related to IoT. It is widely used because it allows multiple different devices (e.g., from different manufacturers) to speak a common language.

It defines a network layer (NWK) above the ones defined by 802.15.4 providing self organizing multi-hop network capability, distance vector routing and neighbor discovery in different topologies including star, mesh and cluster tree topologies which are the same as illustrated in Figure 2.2. There are three types of nodes: Zigbee coordinator, Zigbee router and Zigbee end device. In regards to 802.15.4, Zigbee coordinators are similar to PAN coordinators that manage the network and store information like security keys, Zigbee routers to FFDs that relay information from other devices and Zigbee end devices are generally RFDs that are resources constrained devices able to communicate with coordinators and routers but not with others of their own kind.

It also defines an application layer divided in three sub-layers. The first one, the Application Support Sub-Layer, is the interface between Zigbee device objects and the network layer providing a set of services. The second one, the Zigbee Device Object (ZDO), performs control and management of application objects. The third one, the Application Framework, is an execution environment for hosting the application objects holding user/Zigbee applications.

Z-Wave

Z-Wave [31] is another proprietary protocol quite similar to Zigbee. It is a technology that also uses mesh networks, allowing nodes to act as repeaters and allowing multiples routes for a node to communicate with another one via low-energy Radio Frequency (RF). Data can also flow in the network to a controller, i.e., a gateway connected to the Internet (in the case were it needs to join a destination outside the Z-Wave network) and that is managing communications between the network's actors. Z-Wave is mostly used in home automation.

[1] `https://www.zigbee.org/`

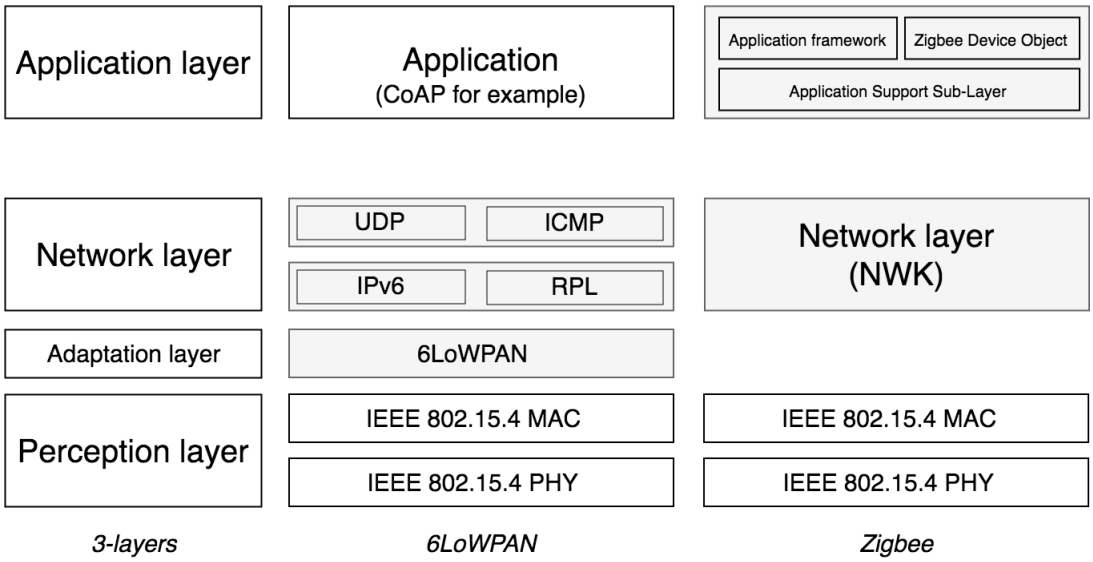

Figure 2.4: 6LoWPAN and Zigbee stacks on top of 802.15.4 standard.

LoRa and LoRaWAN

Very famous in the field of smart cities, Long Range (LoRa) technology [34] is a standard radio modulation technology for Low Power Wide Area Network (LPWAN) providing efficient exchange over long range. Long Range Wide Area Network (LoRaWAN) [34] is a protocol built on top of the LoRa technology, defining the MAC layer, and developed by the LoRa Alliance[2]. The specificity of this protocol stack is that it can very well manage power consumption of devices (up to twenty years [54]) at long range and contained within a small volume at a low cost.

The LoRa Alliance protocol architecture follows a "star of star" topology as depicted in Figure 2.5 as several end nodes communicate with one or more gateways and these gateways communicate with a single network server. Nodes can be monitoring devices or basic devices often low powered with sensors and transceivers embedded. They use the LoRa RF technology to communicate data with a network server through a gateway which, in its turn, uses standard Internet Protocol (IP) connections. The network server parses the data sent from the nodes and sends the payload to an application server.

2.2.4 M2M communication

Machine-to-Machine (M2M) communications refer to communications occurring at the application layer, mainly between the smart things.

MQTT

The lightweight MQTT protocol [41] provides a way for resource constrained network clients to exchange messages in a simple subscribe/publish model. In IoT, it is used for M2M communication. We can put it in the same group of protocols as Hypertext Transfer Protocol (HTTP) but MQTT turns out to be a better option for IoT as it has a way smaller packet overhead. Today, MQTT acronym is no longer considered as such since there was a shift of the primary focus of the protocol from proprietary embedded systems to IoT use cases [25].

There are two types of components that we can find within MQTT: several clients and a broker. Furthermore, we distinguish two different kinds of clients: subscribers and publishers.

[2] `https://www.lora-alliance.org/`

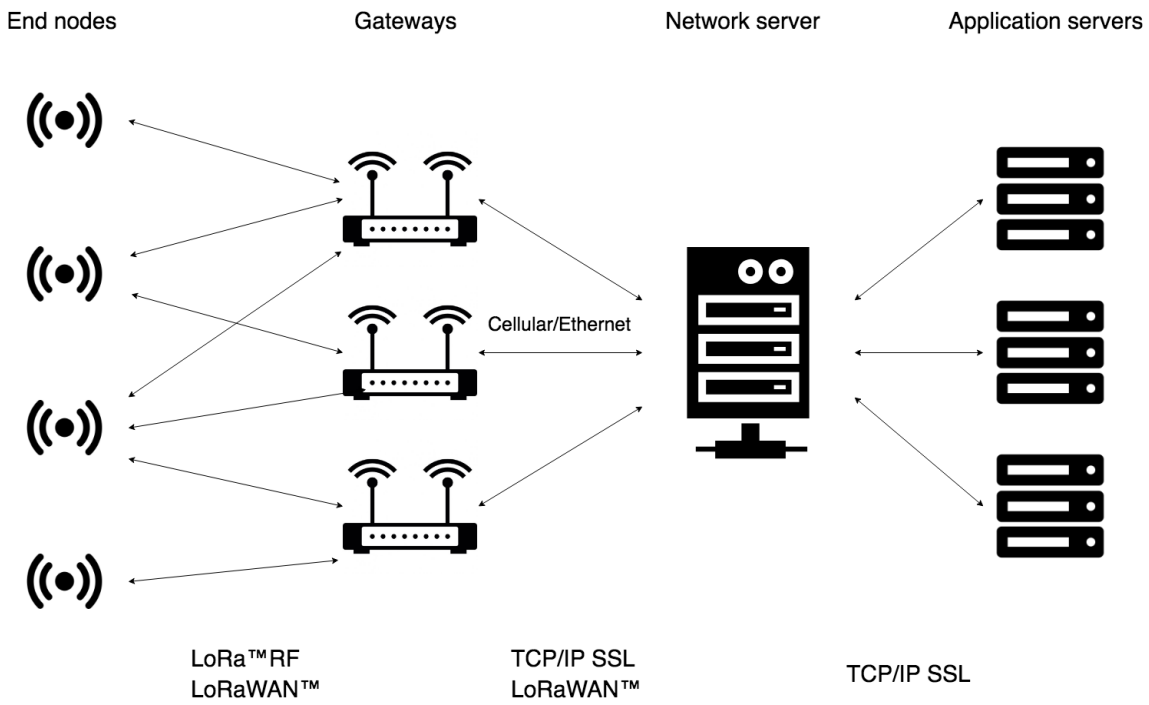

Figure 2.5: Star of star topology followed by LoRaWAN protocol.

Subscribers register for a specific topic (following a file path structure) for which they want to receive messages while publishers push messages to a specific topic. The broker is a server that acts as an intermediary between both types of clients, relaying messages from publishers to the right subscribers by doing subject-based filtering and buffering non-received messages to send them later to a client in case of a disconnection. Clients never contact each other directly to the point that they don't even know each other existence and thus exchange in an asynchronous way.

This protocol is explained with greater details in Section 3.1 as it constitutes the foundation of the dionaea honeypot's extension.

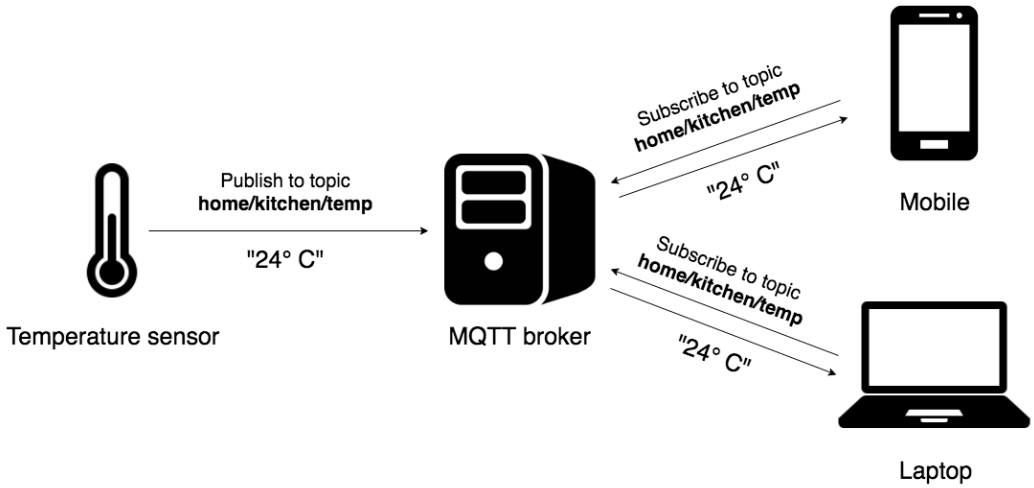

Figure 2.6: MQTT protocol with a subscribe/publish model.

2.3 Vulnerabilities in IoT technologies

In this section, we present vulnerabilities observed in IoT technologies discussed earlier by grouping them in categories of well-known attacks.

2.3.1 Denial-of-Service

Denial-of-Service (DoS) attacks can be executed in many ways in IoT. They aim at making a system unavailable to legitimate users, forcing it to work under reduced performances or even to not work at all.

RFID jamming

One of the greatest problem with regard to RFID technology is *jamming attacks* [55]. Legitimate traffic is jammed by overwhelming the frequencies used by the protocol. Simply creating interference by sending random bits to prevent the tag from communicating with the reader allows to impact availability as received information can be deformed. Sending bits at a high-rate also impacts availability as it can exceed the maximum rate allowed on a communication channel.

6LoWPAN fragmentation attack

6LoWPAN presents several vulnerabilities allowing an attacker to perform a DoS attack. The adaptation layer of 6LoWPAN, as said earlier, does some fragmentation and compression of IPv6 packets to make them usable in a 802.15.4 network.

Fragmentation attacks are possible from an attacker because a receiver is not able to verify if the source of two fragments belongs to the same IPv6 packet at the 6LoWPAN layer [26]. The receiver is thus left unable to distinguish legitimate fragments from spoofed ones.

In the case where the attacker sends a spoofed fragment for every packet, having a random payload and a fragment header that links it to that packet, 6LoWPAN would be unable to decide which fragment is the one to include in the packet reassembly. The entire packet would be considered as corrupted because of the complexity of dealing with such a situation and would be dropped [26], causing a loss of information and thus a possible malfunction on the receiver (and possibly the entire network).

RPL network congestion

DIS messages in RPL can be used to perform a *HELLO flood attack* [38]. From the moment where an attacker managed to insert its malicious node inside the RPL network (by advertising a strong signal power for example), it can program it such that it broadcasts DIS messages to its neighborhood, requiring them to send DIO messages.

Of course the trickle timer can limit the DIO messages sent by the nodes but it can be reset for every node by the attacker that sends bad messages, like messages that announce a change in the network topology or a new DODAG version.

The effect of that can be a congestion of network links and unnecessary use of battery making the nodes running out of power.

LoRaWAN selective jamming

For some protocols like LoRaWAN, *jamming attacks* can be avoided when they are used continuously [45]. This is where a more advanced type of jamming comes in, *selective jamming attacks*.

This type of jamming selects specific devices or messages to jam based on information observed in packets. In LoRaWAN, the first four bytes of the *FrameHeader* which is included in the non encrypted MAC payload contain the end-device address of the sending node. These four bytes can be used to selectively jam specific devices, making the task harder for a network administrator to know if the device is really being jammed or if there is a technical problem as other devices are not in that case [1].

2.3.2 Cryptographic key compromising

One of the challenges of IoT is connecting the networks of IoT nodes to the Internet, allowing them to communicate via a secure channel that protects the flow of information. When technologies use this kind of protection, there is a need to first exchange cryptographic keys that will make possible the future communications and therefore, a way to manage these keys efficiently such that communications remain safe as long as the system is running. However, even if manufacturers that embeds these technologies in their product say they are secure, we can observe some vulnerabilities in the keys exchange that can compromise any security measure regarding cryptography.

BLE vulnerable pairing modes

Although its session encryption is relatively secure, BLE presents vulnerabilities in its pairing methods Just Works™ and 6-digit pin [51], which are two methods used to exchange cryptographic keys.

When two devices try to pair up, they need to share a common secret known as a Long Term Key (LTK), a key that will be reused for all future sessions. To derive that common secret, both devices agree on a 128 bits AES Temporary Key (TK) which depends on the pairing method. This value is then used to calculate a "confirm" value. All values used in the exchange, except the TK, are transferred in plain text. The value of the TK for the Just Works™ pairing method is 0 while the 6-digit pin pairing method one is a value between 0 and 999,999 padded to 128 bits.

A simple brute force algorithm can then be used to guess the value of the TK just by calculating the confirm value for every possible TK value between 0 and 999,999. Once the confirm is found, both devices keep the exchange going by establishing a Short Term Key (STK) and finally a LTK.

Cracking the TK allows to recover the STK as STK messages are encrypted with the TK. Then, having the STK allows to recover the LTK as STK is used to establish a session over which the LTK is exchanged. Finally, having the LTK makes possible to decrypt any future communication between both devices.

`crackle`[3] is a tool that makes this attack possible.

Zigbee Over-The-Air key exchange

Zigbee security keys can be distributed either Over-The-Air (OTA) or by pre-installing them in the devices. OTA alternative is usually used as it makes it easier for frequent key rotation [66].

[3] `https://github.com/mikeryan/crackle/`

Combined with the standard security level, it presents a serious vulnerability as the key is transmitted in plain text [62]. Even though it's only sent once, it's still possible for an attacker to sniff, at the right time, the packet containing that key in plaint text and then decrypt each packet he can listen to that are transiting in the network.

Also, when keys are stored in the devices, an attacker can simply read the content of it if he has physical access to the device and there is no security mechanisms in place to counter that if the attacker also has the right security software in his possession [28].

`KillerBee`[4] [66] is a hacker-friendly interface allowing to attack Zigbee and 802.15.4 networks. Particularly, it includes `zbdsniff` which is a OTA cryptographic key sniffer.

Z-Wave pairing mode downgrade

Z-Wave older pairing mode *S0* had a serious vulnerability as the network key was encrypted with a key only composed of zeros and exchanged as such between nodes [18]. The company adapted the exchange mechanism by replacing *S0* with another mechanism, *S2*. However, researchers from Pen Test Partners discovered a weakness in this more recent paring mode allowing a *downgrade attack*, switching the paring mode from *S2* to *S0* [61].

It is specified that devices and controllers are backwards compatible concerning the pairing. It means that a controller supporting *S2* will use the pairing method *S0* if the device only supports *S0*. Also, a *S0* controller will pair without using encryption for exchanging the key if the device does not support *S0*.

S0 and *S2* pairing modes are illustrated in Figure 2.7. Basically, we can see that the flow is quite similar in both modes until "*Security Scheme Get*" and "*KEX get*", meaning that the information allowing a downgrade must be in the "*Node info*" packet.

In fact, this packet contains several information including a `Command classes` field that has a value `COMMAND_CLASS_SECURITY_2` if the device supports *S2* mode. The node info packet is not encrypted so an attacker intercepting it would be able to remove the `COMMAND_CLASS_SECURITY_2` field before sending it to the controller. At this point, the controller considers that the device does not support *S2* and uses *S0* pairing mode. The attacker is now able to intercept the key during the exchange, decrypt it as he knows that *S0* mode encrypts it with a key composed of zeros, and then command the device.

2.3.3 Violation of privacy

Attacks on privacy are characterized by taking over information that seems innocuous and using it afterwards to disclose details about an object or an individual. In other words, data is disclosed to unauthorized individuals.

RFID tracking and inventorying

RFID poses some serious concerns about privacy as it is a wireless and silent technology which means that a tag can be read without the knowledge of its owner. In some cases, RFID tags do not make a difference between legitimate requests coming from the deployer of the RFID tag and illegitimate requests coming from anyone else [2]. *Tracking* is then possible in this case [55]. It involves developing an itinerary of a tag, collecting information through signals than can be dangerous or embarrassing for someone, like a location.

[4] `https://github.com/riverloopsec/killerbee`

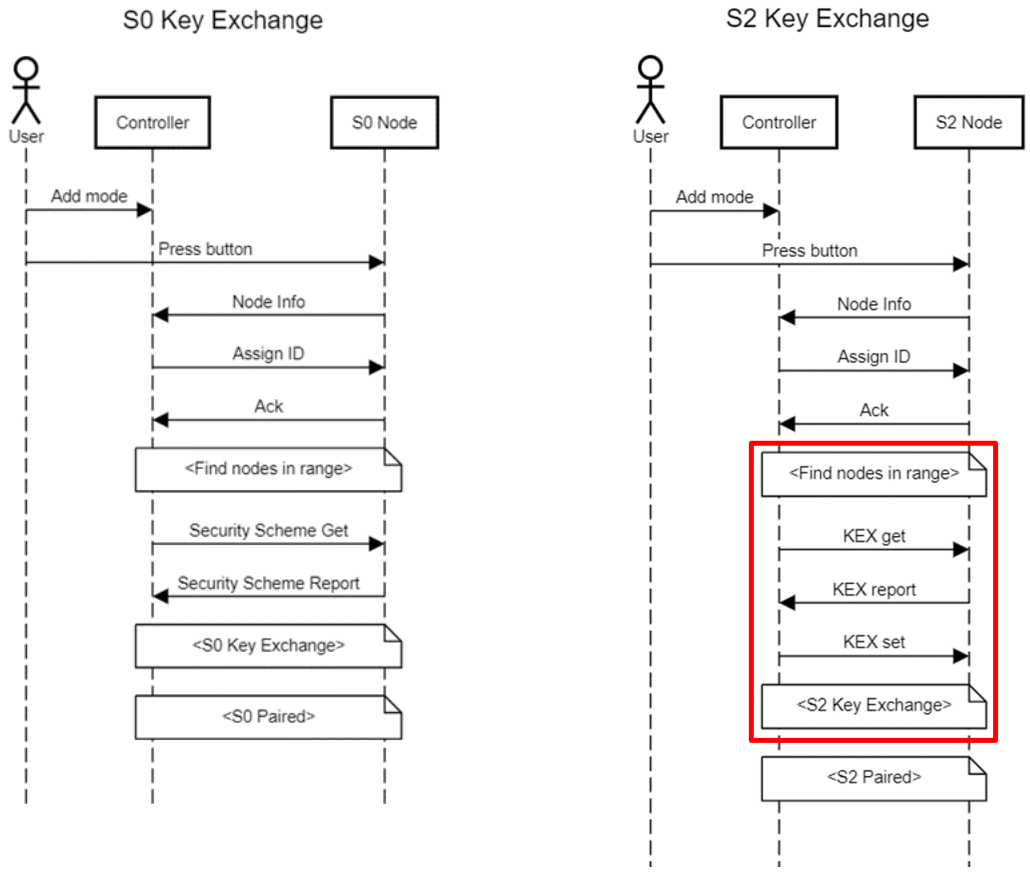

Figure 2.7: S0 and S2 Z-Wave paring modes [61].

Inventorying also occurs [55]. Electronic Product Code (EPC) standard, for example, contains information about a specific item like its manufacturer. A person having an EPC reader and getting close enough to a client at the exit of a shop could determine the customer's habits and inventory them based on labels that are on products working with RFID technology.

RPL wormhole

RPL is subject to privacy issues because of *wormhole attacks* [38]. These attacks are characterized by the presence of two malicious nodes in a network. These two nodes maintain a connection between them and one of the two nodes tunnels the packets it receives to the other node in order to observe the information that it contains in the case where there is no encryption (see Figure 2.8). This is exactly the way it is done in RPL. Wormhole attacks are considered very dangerous as they can easily be launched without any prior knowledge of the network.

Smart applications information leakage

Smart home applications are brought to sometimes leak sensitive information they are using. Earlence Fernandes et al. [17] demonstrated that smart application frameworks like Smart home API[5] allowing third party to develop their own smart application can expose all the sensitive data to an attacker having access to a device. The reason is that they allow permission-based access control to access sensitive data but do not provide any control on how the data is used afterwards.

[5] `https://www.smartthings.com/`

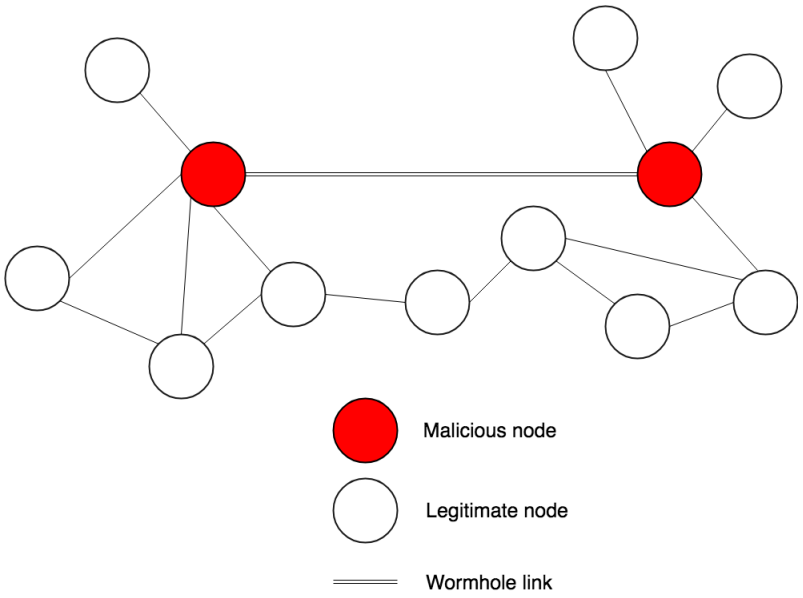

Figure 2.8: Two malicious nodes in a network forming a wormhole.

Backdoor in IoT end-devices

Backdoors are a type of malware avoiding authentication procedures to gain access to a system [27]. They are used for different malicious activities, one of them being the violation of privacy.

Backdoors can be set up intentionally or unintentionally by products manufacturers. The DblTek branded devices case was discovered in March 2017 by Trustwave [30] [33] exposing that almost all devices produced by the VoIP specialist had a remotely exploitable issue in the Telnet administrative interface by presenting a flawed challenge response mechanism. The vulnerability originates from the fact that it was built as a debugging aid according to the manufacturer. Nonetheless, such backdoor can be used by a remote attacker to gain full access to a device.

Backdoors are very difficult to remove. Scanners can be used for detection of malware signatures but it is further complicated as many IoT applications are relying on third-party work which can also contained built-in backdoors [27].

2.3.4 Impersonation

In an impersonation attack, the goal of an attacker consists in faking the identity of a legitimate part involved in a communication with another legitimate part and impersonating him [20].

NFC relay attack

A *man-in-the-middle attack* consists in an attacker inserting himself between two parties that are communicating. He receives message meant for someone else, modifies them and passes them on [60].

A *relay attack* is a subset of the man-in-the-middle attack where the attacker does not attempt to modify the messages and passes them between the two parties [60]. It was first introduced in 1976 by John Conway [12], explaining how a player that has zero knowledge of the chess rules could win to a Grandmaster.

Relay attacks are a kind of attack that are impossible to completely counter in NFC technology. Contactless payment system is a NFC solution where such attack can be used [60]. An attacker

needs to have in his possession a pair of devices that relay the communications between a payment terminal and a contactless card. Once the payment needs to be made, the attacker holds his first device close to the reader then, an accomplice holds the second device close to somebody else's contactless payment card that could be in their wallet for example. The first device relays the instructions coming from the reader to perform the payment to the second device, which sends the instructions to the contactless payment card. The result is the same as if the contactless card was in direct communication with the reader, allowing an attacker to purchase an item without using his own money.

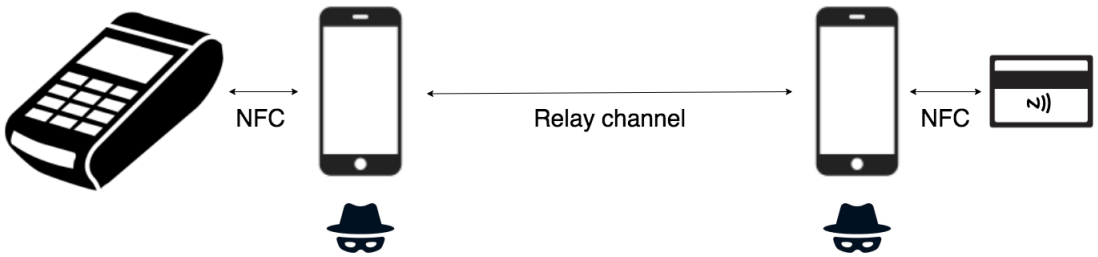

Figure 2.9: Relay attack against NFC payment system.

Z-Wave Sybil attack with a fake controller

Loïc Rouch et al. demonstrated how a *Sybil attack* is possible against Z-Wave by using off-the-shelf hardware and without having any insights of the network's topology or without compromising any original devices [50]. The attack consists in using a universal adversary Z-Wave controller, the coordinator of a Z-Wave network, by reprogramming a USB stick controller.

Manufacturers building products based on an official stack like the Z-Wave one often add their own functionalities in addition to the ones provided by the stack itself. Several manufacturers provide a backup and restore functionality in order to ease the replacement of a controller and the re-building of the network as it can be a tedious task requiring physical interactions from the user [50]. This backup and restore functionality is necessary for this Sybil attack to be performed. The attack scheme is the following:

1. Associate the adversary controller with all possible nodes.
2. Capture the HomeID of the targeted network to be used by the adversary controller.
3. Copy the captured HomeID to the adversary controller.
4. Restart the adversary controller.

HomeID is set during controller manufacturing and cannot be changed manually by a user. It is exchanged by all devices in all their messages and changes its value to a new random one when the controller is reset. By using a receiver and a program able to decode Z-Wave frames like `Waving-Z` [6], it is possible to get it.

Once the HomeID is captured, it needs to be assigned to the adversary controller but Z-Wave protocol makes it impossible for a user to manually doing this because of security issues. However, the backup and restore feature of the adversary controller can be used to clone the captured HomeID. When this functionality is triggered on an adversary computer, it sends a command to

[6] `https://github.com/baol/waving-z`

the adversary controller with a HomeID to be set and saved, allowing to keep already registered nodes in the network.

Then, the adversary controller needs to be able to send commands to the nodes of the target network but this is not yet possible as they aren't known by the controller: a controller cannot send messages to nodes with which it is not paired. Each device is uniquely identified with a NodeID, the controller having a NodeID value of 1. During a pairing sequence, the controller chooses a NodeID for each device. There can only be 232 nodes in a Z-wave network as some NodeIDs are reserved. Instead of listening for every node in the network to send information over the network and getting their NodeID, the idea is to use a single adversary node and to register it as multiple devices. This latter is assigned a NodeID, beginning at 2, paired with the adversary controller such that the latter keeps it in its memory and finally resets to make it appear as a new node. The same steps are then performed with the next NodeID. As pairing 232 times the same would be tedious, partial restore functionality which only changes the HomeID is used.

Finally, only a restart of the controller is needed as Z-Wave auto-discovery feature will automatically probes registered devices in the adversary controller's memory that will respond if they are available. The attacker is now able to communicate with the devices at will.

LoRa malicious Microcontroller Unit

In addition to the encoding and transmission of data over a LoRaWAN network, a transceiver module can be used to perform encryption of the information sent by the Microcontroller Unit (MCU) over UART, for example [16].

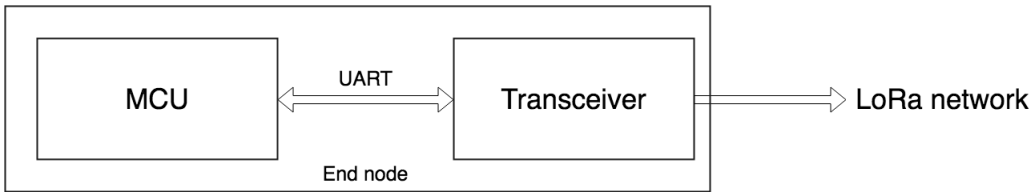

Figure 2.10: Constitution of a LoRa end node.

In this case, the MCU is unaware of the encryption keys used by the LoRa network. If the physical location of a device is revealed to an attacker, he would be able to replace the MCU with one of its own, sending messages on behalf of the node.

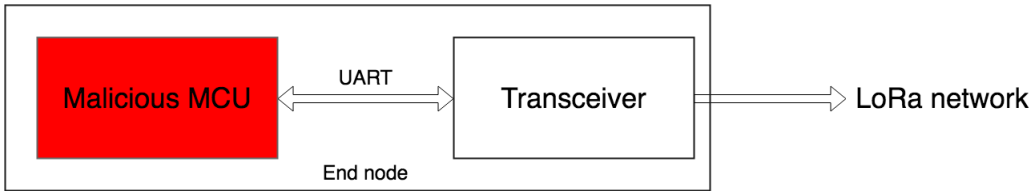

Figure 2.11: Compromised LoRa end node with a malicious MCU.

2.3.5 Botnets

Botnets are probably the biggest threat that targets IoT in terms of device infection scale. Basically, a botnet consists of a collection of compromised devices, often referred to as zombies, that are under remote control of a central server managed by an attacker.

Mirai

First identified in August 2016 by the whitehat security research group MalwareMustDie [35], Mirai turned out to be the originator of one of the first Distributed Denial-of-Service (DDoS) attacks in history using IoT devices, taking down big sites including Twitter, Netflix, Reddit, and GitHub [64]. Its code targets poorly secured IoT devices having weak/default `Telnet` credentials.

More specifically, Mirai belongs to the worm family of malwares. We can distinguish several phases in its process [58]:

1. **Preparing the Attack**: this first phase consists in identifying potential new zombies. It does so by probing pseudorandom IP addresses on Transmission Control Protocol (TCP) ports 23 and 2323 (avoiding a list of specific IP addresses like the one of the Department of Defense). Once a potential victim has been identified, it tries a brute force authentication by using a dictionary of known credentials. In case of a successful login, the victim's IP is sent to a report server.

2. **Infecting the devices**: the report server that previously received the IP of a vulnerable device forwards this information to a malware loader. This latter loads an executable onto the device that is specific to its architecture. The device is now part of the botnet, scanning in its turn for other potential victims.

3. **Attacking the victim**: the botmaster uses a command & control server to which he issues an attack to be executed. This command & control server relay the attack to all the nodes constituting the botnet, telling them to send packets to the victim as quick as possible.

4. **Covering tracks**: Mirai deletes itself from the running processes found in the file system once it is running but it also tries to protect itself by deleting every other processes considered as a threat, i.e. "competing" malwares and other processes that use `Secure Shell (SSH)`, `Telnet` and `HTTP` ports.

Mirai's code was published online, which immediately allowed the creation of new and more advanced botnets.

BrickerBot

BrickerBot is getting very close to Mirai as it uses `Telnet` brute force authentication as well. As the name of the botnet implies, BrickerBot's goal is to "brick" the device, rendering it permanently unusable and requiring a replacement or a reboot of the hardware [53].

The major difference is in the actions that are taken once a device is infected: upon successful access to the device, a Permanent Denial-of-Service (PDoS) is performed by issuing a series of Linux specific commands leading to a deletion of all the files on the device, a corrupted storage or even, a disruption in the device's performances [53].

Reaper/IoTroop

Called "Reaper" or "IoTroop", this botnet also borrowed some code from Mirai. Instead of trying to authenticate into the device by using weak credentials, it focuses only on exploiting vulnerabilities of some IoT devices like home routers and IP cameras [49].

The main infrastructure looks a lot like Mirai's one but there are some differences that we can spot [49]:

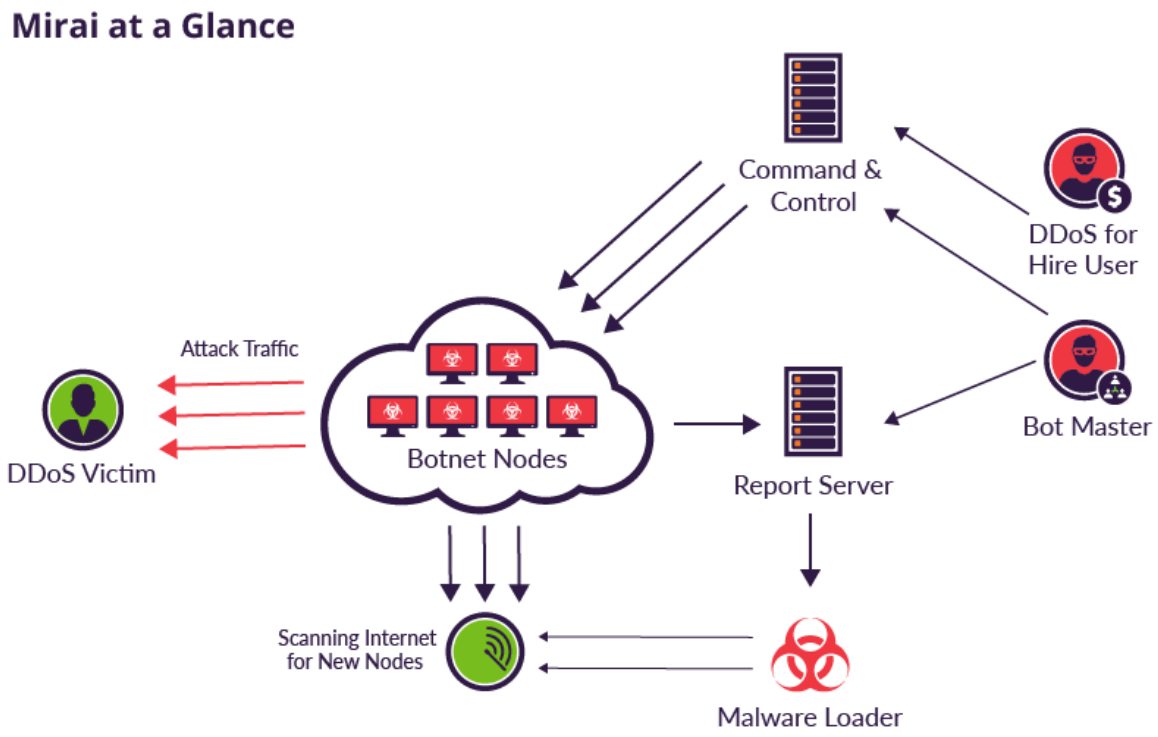

Figure 2.12: Mirai botnet architecture [58].

1. The command & control server has been completely redesigned, using `LUA`[7] scripting to easily change malicious code on the fly.

2. The command & control communication protocol was also changed along with the server. This functionality is brand new and unique to IoTroop itself.

3. The brute forcing attack present in Mirai using a dictionary of well known credentials has been replaced by a vulnerability scan targeting specific IoT devices presenting vulnerabilities.

4. IoTroop does not contain a DDoS functionality. It stayed inactive for a long period until it was suspected of targeting the financial sector in January 2018 [40].

Hide 'N Seek

Hide 'N Seek botnet brought, in its turn, a set of improvements based on Mirai's code. First spotted on on January 2018 [8], it is the first botnet using custom-built Peer-to-Peer (P2P) communication instead of using a command & control server to execute malicious commands. The bot performs brute force authentication as well as web exploitation against devices that present the same exploit as Reaper/IoTroop. Hide 'N Seek has three main modules [36]:

1. **Scanner**: it generates a random set of IP addresses in the search of potential victims. The difference with Mirai is about the ports that are being scanned. Hide 'N Seek scans ports 80, 8080, 2323, 9527 and 23. Once the connection is established with the device, it tries a brute-force authentication via `Telnet` using a dictionary of credentials.

2. **Killer module**: it kills processes running on `Telnet`, `SSH` and `HTTP` as well as other botnet malwares.

[7] https://www.lua.org/about.html

3. **Peer connection function**: it waits for incoming connection with peers to execute/relay an attack command.

As previous botnets, Hide 'N Seek was not able to achieve persistence which means that a reboot would basically bring the device back to its original state [8]. However, findings in May 2018 stated that the botnet was identified as having ten different binaries for various architectures and was now able to add itself in the initialization script of the devices in order to achieve persistence [7].

Chapter 3

Extension requirements

This chapter explains, in more details, the two technologies necessary for the realization of our second big objective which is the extension of an existing honeypot.

The first section addresses a deeper look into MQTT in order to understand the functionalities that were added. The second section talks in a general way about honeypots and also explains the major cogs of dionaea honeypot.

3.1 Deeper look into MQTT

Among the IoT technologies one seemed particularly interesting to us regarding the intention to develop a honeypot. The boom of IoT technologies, the poor security of these systems and the growing popularity of MQTT make this protocol a perfect candidate for potential hackers. Indeed, this protocol is appreciated for its lightweight, maturity, stability and reliability. Figure 3.1[1] clearly demonstrates the popularity of MQTT compared to other similar protocols.
This section lays the MQTT v3.1.1 essentials [24].

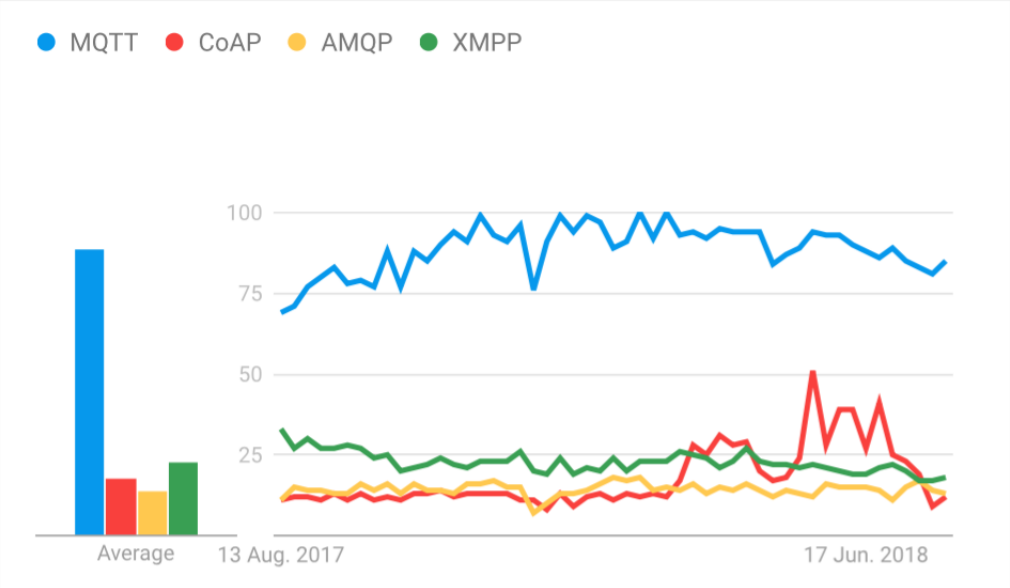

Figure 3.1: Worldwide last 12 months Google Search

[1] https://trends.google.com/trends/

3.1.1 Quality of Service

Let us first have a look at Quality of Service (QoS) since this core feature is involved in almost every aspect of the MQTT protocol, present at connection establishment, subscription and publication.

The QoS level is an agreement between the client and the broker regarding the guarantee of a message delivery when it is published in the network (from a client to the broker or the forwarding from the broker to the subscribing clients).
There are 3 levels of QoS in MQTT:

- At most once (0)
- At least once (1)
- Exactly once (2)

This feature allows the user to choose the QoS that matches the network reliability and the application purpose, this makes communication in unreliable network much easier. A closer look at each QoS level follows.

QoS 0

This is the simplest way of message transmission. Often called "fire and forget", QoS 0 does not give any guarantee of message delivery. The receiver does not acknowledge the reception of the message. Thus, there is no extra guarantee than the one provided by the underlying TCP protocol.

QoS 1

With QoS 1, the message is guaranteed to be delivered at least once. The sender keeps it in memory until it gets a `PUBACK` packet in response which acknowledges the reception of the previous publication. It is possible for a message to be delivered several times, for instance if the acknowledgment gets lost the sender will eventually send the message again.

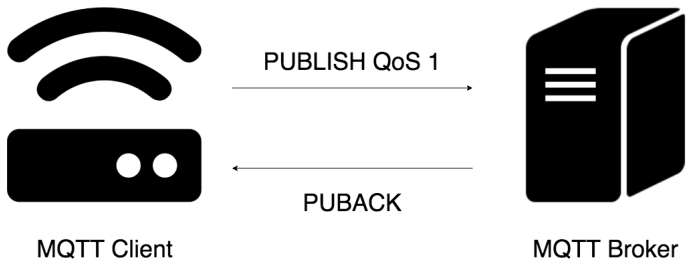

Figure 3.2: QoS 1 Acknowledgment

QoS 2

QoS 2 is the safest and the slowest transmission mode of MQTT. This level guarantees (with a four-way handshake) that the message is delivered exactly once.

When a client sends a message with QoS 2, it is stored until the reception of a `PUBREC` packet. Then the client sends and stores a `PUBREL` message until reception of the corresponding `PUBCOMP`.

Regarding receiver side, after reception of the first `PUBLISH` message, it responds with a `PUBREC` which is kept in memory and only discarded when the responding `PUBREL` arrives. The latter is acknowledged with a `PUBCOMP`.

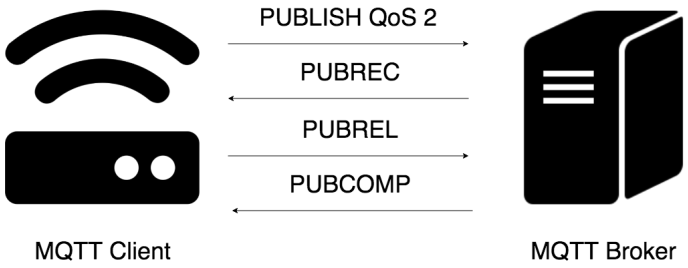

Figure 3.3: QoS 2 Acknowledgment

Two sides delivery

It is important to note that the end-to-end communication in MQTT is performed by two consecutive transmissions: from the publisher to the broker and from the broker to the subscribers. Yet, the QoS level of each transmission might be different.

Indeed, subscribers specify the highest QoS level they accept and it might be lower than the one used by the publisher of the message. In that case, the broker has to downgrade the QoS level to match the requirement of the subscriber, but still communicates with the publisher with a higher QoS level. Thus the QoS level might differ from one side of the delivery to the other.

3.1.2 Connection establishment

The protocol is built upon `Transmission Control Protocol/Internet Protocol (TCP/IP)`.

To establish a connection, a client sends a `CONNECT` message to the broker, which responds with a `CONNACK` message. A `CONNECT` packet is made of numerous fields : `clientId`, `cleanSession`, `username`, `password`, `lastWillTopic`, `lasWillQos`, `lastWillMessage`, `lastWillRetain` and `keepAlive`.

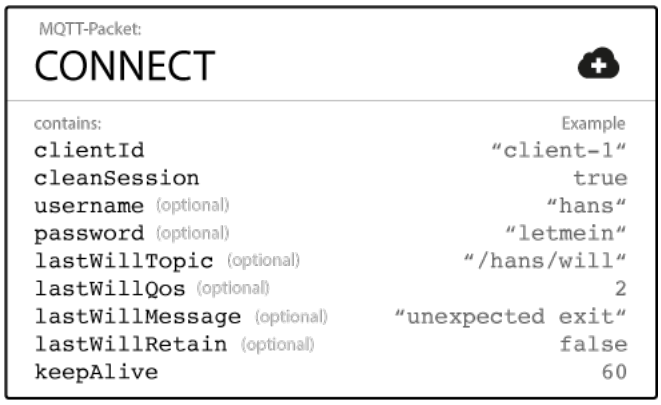

Figure 3.4: `CONNECT` Packet [24]

clientId The client identifier identifies each client connected to the broker.

cleanSession This flag specifies whether the client wants its session to be persistent or not. If this is the case (`cleanSession = False`), the broker stores each subscriptions and missed messages for the client. In case of a sudden disconnection for instance, the broker keeps in memory all the messages intended for the client until it reconnects.

username/password Can be use for authentication and authorization, the use of these fields is let up to the broker designer. However, nothing is encrypted at MQTT level, the password is sent in plain text.

lastWill This is part of the so called "Last Will and Testament (LWT)" feature of MQTT. The four lastWill fields are used to notify other clients in case of an ungraceful disconnection. The message is sent by the broker to specified topics with the desired QoS and Retain flags (explained in the following sections).

keepAlive Is a time interval (in seconds) sent by the client to specify how long the broker and the client can stay without sending any message. The client sends regular `PINGREQ` (Ping Request) messages that are acknowledged by the broker. If the client doesn't receive any `PINGRESP` (Ping Response) or the broker doesn't receive any `PINGREQ` within the time interval specified by the `keepAlive` at connection, they know that their peer is not available anymore.

`CONNACK` packets are only made of two fields : `sessionPresent` and `returnCode`.

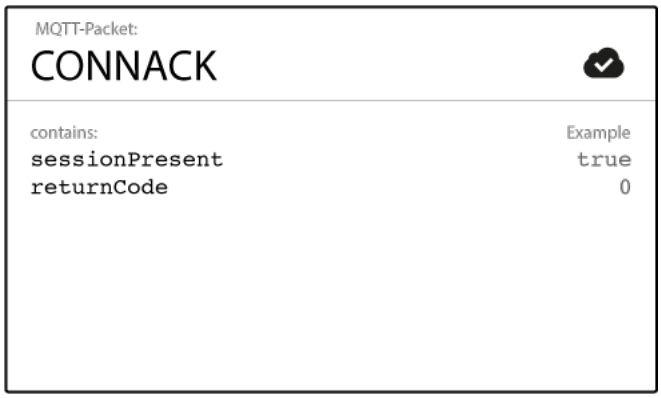

Figure 3.5: `CONNACK` Packet [24]

sessionPresent Is a flag set to `True` if a session was already established before for that client and is still available.

returnCode Is 0 in case of a successful connection establishment. Codes 1 to 5 indicate a connection refused for various reasons.

3.1.3 Subscribe, Unsubscribe & Publish

Subscribe

To receive messages on topics of interest, the client sends a `SUBSCRIBE` message to the broker. Such a packet contains a unique `packetId` and a list of subscriptions: a list of topics and associated QoS (see 3.1.1).

Topics look literally like a file path having different levels with each level separated by a forward slash. The broker accepts any topic without any prior initialization as long as it has

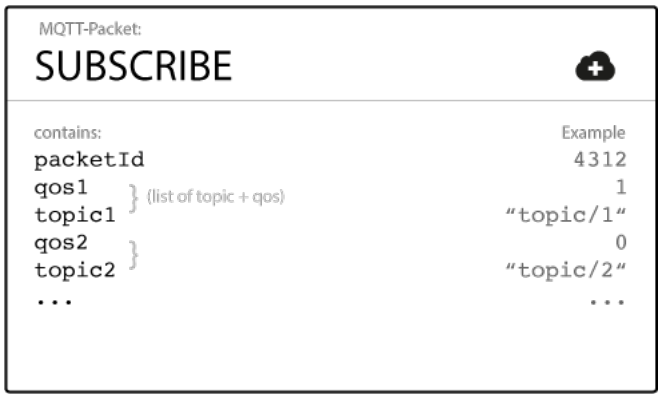

Figure 3.6: `SUBSCRIBE` Packet [24]

the correct form. Subscription topics can include wild cards `+` (single-level) and `#` (multi-level) in order to subscribe to multiple topics at once. Single-level wild cards replace one topic level, allowing to match any registered topic that has an arbitrary string instead of the wild card. Multi-level wild cards can only be at the last level of a subscribe topic, allowing to match any registered topics that have the levels before the `#` level as a prefix.

For example, a message published to the topic `home/kitchen/temp` will be received by subscribers to `home/+/temp`, `home/kitchen/#` or even `#`.

The broker must respond with a `SUBACK` message, which contains the `packetId` and a list of return codes. There is one return code by topic in the list of subscriptions of the previous `SUBSCRIBE`, it is equal to the QoS set for the subscription (0, 1 or 2) or 128 in case of a failure.

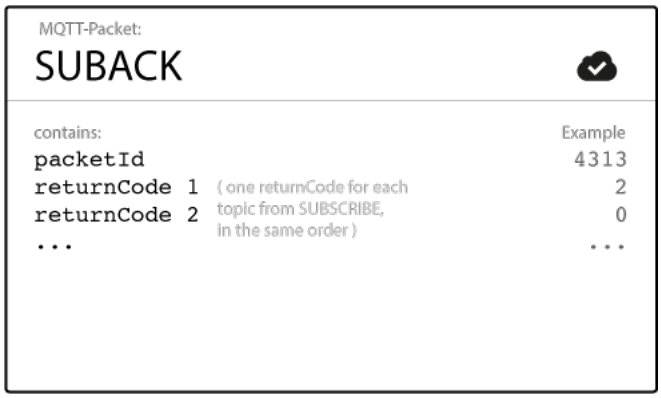

Figure 3.7: `SUBACK` Packet [24]

Unsubscribe

The unsubscription mechanism is quite straight-forward and looks a lot like the subscription. `UNSUBSCRIBE` packets are made of a `packetId` and a list of topics to unsubscribe to. The broker acknowledges by a simple `UNSUBACK` packet containing only the corresponding `packetId`.

Publish

`PUBLISH` messages are used both by clients (typically sensor devices) to push data on a topic and by the broker to forward the data to every subscribers. The `PUBLISH` packets contain six mandatory fields : `packetId`, `topicName`, `qos`, `retainFlag`, `payload` and `dupFlag`.

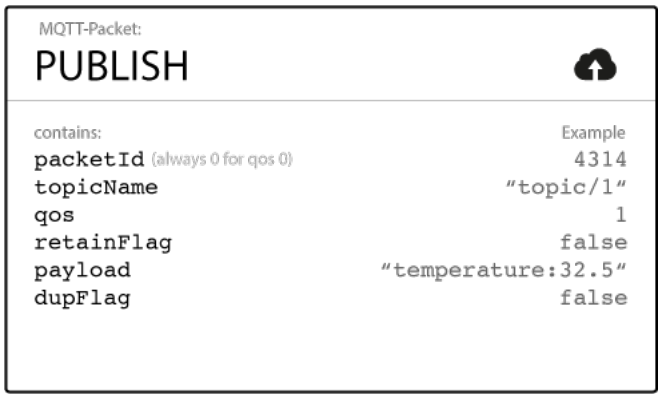

Figure 3.8: `PUBLISH` Packet [24]

packetId The packet identifier uniquely identifies a message. This is only useful for QoS 1 and 2 thus it should always be set to 0 for QoS 0.

topicName The topic where to push the data. The format should respect the one described in the Subscribe section 3.1.3.

qos The QoS level of the message. See section 3.1.1.

retainFlag One message can be retained for each topics so that a new subscriber to a topic automatically receives this message. If the `RETAIN` Flag of a `PUBLISH` packet is set to `True`, the previous (if any) retained message is overwritten by this one.

payload The payload contains the actual content of the message. MQTT being data-agnostic, the payload may be structured anyhow and it is possible to send text, images, encrypted data and every data in binary.

dupFlag The `DUP` Flag indicates that the message is resent because it has not been acknowledged the first time. This is only relevant for QoS greater than 0.

Once the broker receives a `PUBLISH` message, it transmits this message to all the subscribers of the topic (potentially with a downgraded QoS) and acknowledges the publisher according to the QoS level specified.

All the Publish Acknowledgement packets (`PUBACK`, `PUBREC`, `PUBREL` and `PUBCOMP`) have the same structure. They only have one field: the corresponding `packetId`.

3.1.4 MQTT v5

The MQTT version used as part of this study is 3.1.1, however this is not the latest MQTT standard. Indeed, the Technical Committee (TC) responsible for specifying and standardizing MQTT at Oasis released the official MQTT v5 specification late December 2017. On a side note, due to the fact that the code used to specify the protocol version in `CONNECT` packets for MQTT v3.1.1 is 4, there is no 4th version of the protocol, the previous one is indeed 3.1.1.

MQTT v5 improvements

Several years after MQTT's production deployment, enough feedback has been received to improve the protocol. This version is a major update to the existing protocol specification and so is not backward compatible wit v3.1.1 (unlike this version was with v3). It aims to meet longtime

users demands without adding unnecessary complexity and by staying as lightweight as possible.

The Oasis TC set five goals for this new specification [23]:

- Enhancement for scalability and large scale systems;
- Improved error reporting;
- Formalize common patterns including capability discovery and request response;
- Extensibility mechanisms including user properties;
- Performance improvements and support for small clients.

With these goals in mind, some very useful new features have been specified.

For instance, Time to Live (TTL) allows to set the time a retained message or a client session should stay in storage before discard.

Another major improvement (mainly for scalability) is the new *Shared Subscriptions* feature, also known as *Client Load Balancing*. Clients have now the opportunity to group to subscribe to topics so that messages published on the considered topics are distributed amongst the clients of the group. This feature is easy to use, subscribers only have to subscribe to a topic formatted that way: `$share/GroupID/Topic`, and the load will be balanced by the broker.

A very appreciated feature by the developers is the *Reason Code* which is carried by acknowledgment packet in order to give more information about a protocol error.

A new packet type has also been added and is expected to be used a lot in production environments: `AUTH`. It is useful to implement non-trivial authentication mechanisms and to re-authenticate clients without closing the connection.

Finally, the last feature that we will introduce here (even if this update contains many more modifications) is the *`CONNACK` Return Codes* for unsupported features. A lot of MQTT implementations were created by numerous companies but not all of them are completely MQTT specification compatible. It is not rare that some features are missing, mostly QoS 2, retained messages or persistent sessions. In MQTT v5, some new headers may be added to `CONNACK` packets in order for the broker to notify the subscribers that some features are not supported.

Why MQTT v3.1.1

With all the improvements described above, one is now in position to question why focusing on older version of the protocol. The main reason is the timing, at the time of the redaction of this thesis MQTT v5 specification is only seven months old. Really few tools are provided to developers yet and those which are are still under development and testing.

Eclipse Paho, the most common MQTT library, was planning to release MQTT 5 client libraries in `C, Java, Go` and `C++` by the end of June 2018 but they are still under review for the most part (see the version 5 absence on the Paho downloads page[2]).

Even `mqtt.org` does not mention MQTT v5 on its main page.

Thus, we did not consider the version 5 to be 100% ready yet, even more considering that the work on this thesis started months before its redaction. Moreover, working in not (yet) well known environment, benefiting from few resources online would have sorely slowed us.

However, we don't consider our efforts to be vain. We expect the development of MQTT v5 to be slow as v3.1.1 is well established and MQTT is typically used in critical environments where unreliable system is not acceptable, which tends to dampen a little the willingness to try

[2] `https://www.eclipse.org/paho/downloads.php`

new things. Even when the version 5 will seriously tend to take over its predecessor, one can assume that attacks against MQTT v3.1.1 will continue. Attackers are also fond of old versions of protocols since they are often considered as more vulnerable and there are always people running old or even depreciated version of systems somewhere.

3.2 Honeypots

This section focuses on a powerful tool regarding IT systems security : honeypots.

The first part explains what is a honeypot and what kind of such software exist, then some general and IoT specific honeypots are presented. The second part introduces dionaea in details.

3.2.1 What's a honeypot?

Definition

A definition according to Internet Engineering Task Force (IETF) Requests For Comments (RFC) 4949 [57] :

- **Honeypot** : *"A system (e.g., a web server) or system resource (e.g., a file on a server) that is designed to be attractive to potential crackers and intruders, like honey is attractive to bears."*

Indeed, a honeypot is a decoy computer system which aims to trap hackers, track malicious activities and perform analysis. Multiple honeypots set on a network form a honeynet.

We distinguish two kinds of honeypots : research and production. Research honeypots are mainly used by educational institutions and nonprofit organizations in order to gather information about trending hacking methods, discover attackers' motives and learn how to protect against those threats. Production honeypots are used to protect companies and corporations. They are placed inside the production network alongside production servers to improve their overall state of security and understand better the motives of attackers against their own organization.

We can further group honeypots according to their aims and level of interaction:

- **Low-interaction**: Offer limited activity, it generally works by emulating few services or parts of an operating system. The simplicity of low-interaction honeypots makes them easy to deploy and maintain but also easier to identify for the attackers. It is also less risky since the hacker doesn't have access to an operating system which he could take control of to harm others. All in all, these honeypots are useful to collect lots of general data (typically the number of connection attempts, on what port, etc.) at a higher level.
- **High-interaction**: Involve real operating system or emulate a whole operating system and applications. High-interaction honeypots are way more complex, they provide more realistic target more difficult for hackers to identify and they are capable of detecting a higher caliber of attacks. The information gathered is thus more specific, at a lower level. They are also complex to deploy and maintain since they require other technologies to prevent hackers from using the honeypot to launch attacks on other systems.
- **Hybrid**: Combine low and high interaction components. This is a two steps approach, all traffic is first processed by the low interaction component, this allows to cover a broad range of connection attempts. Then, the low interaction component acts as a filter, it only transfers unknown attacks to the high interaction component. The latter deeply analyzes new attacks and so don't waste time on previously seen one. This approach tries to combine the advantages of both methods in addition to provide a protection against hackers trying

to take control of the honeypot to perform DoS attack, since such attempts would be blocked at the first step.

- **Medium-interaction**: Expand low-interaction honeypots in order to offer higher interaction but still less than high-interaction ones. They provide partial implementation of services allowing to collect more information from more complex attacks. But again, the more advanced the less secure the system is.

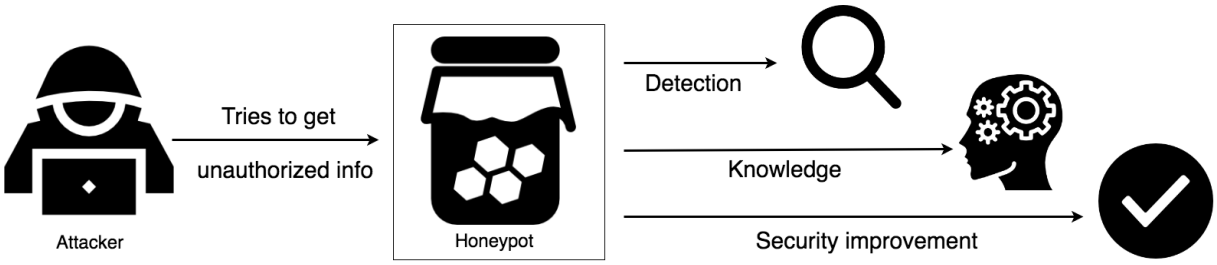

Figure 3.9: Purposes of a honeypot.

3.2.2 Examples of honeypots for IoT

Here are some information gathered about the most relevant honeypots in IoT field, necessary step before choosing the one to extend [9].

Telnet-IoT-honeypot

A really simple low-interaction honeypot running a python Telnet server. The aim of Telnet-IoT-honeypot is just to collect statistics on IoT botnets which spread over insecure default passwords on telnet servers on the Internet.

IoTPOT

IoTPOT [43] is a slightly more advanced Telnet-based medium-interaction honeypot, which supports all the features inherited by the Telnet protocol. The point of this honeypot is to simulate various IoT devices running on different CPU architectures. Their research shown that at that time (2015) they were at least 4 distinct DDoS malware families targeting Telnet-enabled IoT devices.

MTPot

Released in 2016 by Cymmetria Research[3], this open source low-interaction honeypot is designed specially to trap the `Mirai` botnet. It is a simple tool directly based on the source code of `Mirai` to match what the botnet expects. This honeypot has two main goals, firstly to collect a set of IP addresses trying to infect other IoT systems and secondly collect malware samples which are downloaded during the infection.

Honeyd

Honeyd [46] is a very popular open source low-interaction honeypot designed by Niels Provos from University of Michigan in 2003. This is a small daemon able to simulate thousands of virtual host (in Unix systems) at the same time on a single system. Both hosts and routing topology

[3] `http://www.cymmetria.com/`

can be simulated. Hosts can be configured to run arbitrary services on specified ports, it is even possible to adapt their fingerprint so that they appear running specific Operating System (OS) (e.g. IoT architectures). Regarding the routes, they can be configured with latency or packet loss rates so that they seem even more realistic. The obvious goal being to make the hacker think he has access to a whole network. Nevertheless, Honeyd suffer from a major issue : age. Some parts are outdated and nobody in the community seems to update them. The last commit on GitHub dates from December 2013.

Cowrie

Cowrie is an extension of Kippo, the SSH medium-interaction honeypot project closed in 2014. Being the successor of Kippo, Cowrie is itself a honeypot that emulates SSH but this time expanded to Telnet protocol. Other than emulating the Telnet protocol, Cowrie also fixes some well known issues of Kippo and support both SSH File Transfer Protocol (SFTP) and Secure Copy Protocol (SCP) for file upload, among other improvements. The honeypot is used to register brute force attacks aimed at forcing SSH or Telnet passwords (deliberately trivial) and then analyze further actions of the attacker by registering all shell commands inputted by the hacker and malicious files downloaded thanks to SFTP/SCP protocols.

Dionaea

Known as "The Nepenthes [3] Successor", dionaea was developed under The Honeynet Project's 2009 Google Summer of Code (GSoC)[4] by DinoTools[5]. Nepenthes was a low-interaction honeypot designed to especially emulate vulnerabilities that are exploited by botnets. dionaea continues to do more of the same in order to easily gather worm samples (especially directed to Microsoft Windows), but emulates the whole underlying network protocol stack rather than emulating mainly the vulnerabilities. Moreover, it inherits from Nepenthes' modular architecture which works with modules in order to easily add new vulnerabilities (in the case of Nepenhes) or protocols (for dionaea). Indeed, if Nepenthes offered common services like FTP, HTTP, TFTP and MSSQL, Dioneae became quickly very wide with the introduction of numerous protocol modules (such as MangoDB, MySQL, SIP, UPnP or MQTT) to the extent that even IoT devices may be emulated.

SIPHON

The Scalable high-Interaction Physical HONeypot (SIPHON) [22] aims to fill the need to find a way to develop high-interaction honeypots that will attract and fool expert hackers, since most of the honeypots are too easy to detect thus only trap automated attacks (botnets). SIPHON is one of the most advanced technique, it uses actual IoT devices in order to offer advanced services to the attacker (e.g. moving a camera). Thanks to its scalable and distributed architecture, the honeypot deploys over 80 (virtual) devices with a diverse set of IP addresses located in different regions of the world by using only 7 physical IoT devices. Those are connected and replicated to various wormholes. The network traffic is forwarded on an infrastructure of cloud services providers.

Bluepot

Some honeypots focus on one single protocol, this is the case of Bluepot with Bluetooth. This quite complete low-interaction protocol is designed to store any malware sent to it and to interact with well known Bluetooth attacks such as `Blue Bugging` (remotely taking over control of a

[4] `https://summerofcode.withgoogle.com`
[5] `https://www.dinotools.de/en/`

device) or `Blue Snarfing` (remotely reading files from another device). This honeypot is able to simulate several Bluetooth devices and multiple vulnerabilities.

ZigBee Honeypot

Another example is the ZigBee Honeypot [15] which, as its name suggests, emulate a ZigBee gateway. This honeypot captures all the traffic for retrospective analysis with the aim of assessing the presence of ZigBee attack intelligence on a SSH attack vector.

3.2.3 Dionaea

Some of the above mentioned honeypots seem interesting regarding the purpose of this study, each with their advantages and drawbacks. However, dionaea appears to be the most appropriate candidate. Indeed, the module-based architecture makes it easy and secure to extend, it already proposes an elementary MQTT module and even if it was originally designed to emulate a Microsoft system, it is perfectly suitable for IoT systems. Moreover, the project is still active and maintained (several commits a week on GitHub), which is not the case of many honeypots like for instance Honeyd which would have been interesting too but suffers from depreciation. However, the main flaw of dionaea is its poor documentation[6].

This section proposes a deeper look into dionaea's architecture by explaining how it achieves security, a major concern for honeypots, what are the modules and their purposes and how does the logging system works.

Security

All software is likely to contain exploitable bugs and dionaea is a software. In order to minimize the impact, dionaea is able to drop privileges and `chroot`. To be able to run certain actions which require privileges, after having dropped them, the honeypot creates a child process at startup and asks it to run the actions requiring higher privileges. This does not guarantee anything but this is an extra protection.

Modules

dionaea has 4 main modules, among them the `python` module which allows the emulation of many protocols.

- `curl`[7]: The curl module allows to transfer files from and to servers. It is used to download files (e.g. malware samples) as well as submitting them to 3rd parties (online malware analysis services).
- `emu`: dionaea uses `libemu` to detect and evaluate payload sent by the attacker in order to get a copy of the malware. If necessary, `libemu` allows to execute shellcode in the `libemu VM`.
- `pcap`[8]: This module uses the `libpcap` library to detect rejected connection attempts in order to collect information on all connection attempts, successful or not.
- `python`: Python is used as dionaea's embedded scripting language in order to emulate protocols. There it is easy to add new modules dedicated to the emulation of one particular service (e.g. MQTT).

[6] `https://dionaea.readthedocs.io/`
[7] `https://curl.haxx.se`
[8] `https://www.tcpdump.org`

Logging

To keep things simple, dionaea can write logs in a text file. But, dionaea being really chatty, this solution is not scalable, working with text files is not convenient even with good filters. Thus, the honeypot implements an internal communication system way superior than text logging, called incidents. An incident carries an origin (a string), a path and properties (which can be integers, strings or a pointer to a connection) of an attack. Then, incidents pass the information to incident handlers (ihandler). Therefore, information is clustered based on the initial attack when retrieving data from the database and can be written in any suited format. An ihandler can register a path for incidents he wants to get informed about. For instance, the ihandler `log_db_sql` allows to store data in a Structured Query Language (SQL) database, `log_json` converts information in json format, or `log_incident` is used to export incidents in realtime to be processed by external programs via http. On top of local logging system, dionaea can also be configured to send log streams to an XMPP server.

Chapter 4

Dionaea honeypot extension

In this chapter, we describe the changes brought to dionaea's MQTT module in order to turn this module into a medium-interaction one, more realistic and more likely to fool attackers. We compare the old and the new module in respect of the Oasis standard.

The majority of the code we wrote and modified to extend MQTT module in dionaea can be found at: `https://github.com/ysmal/dionaea/tree/master/modules/python/dionaea/mqtt`.

4.1 Previous MQTT module

The MQTT module originally contains, besides the initialization script, two files : `mqtt.py` and `packets.py`. The first one handles MQTT connections, log incoming messages (by creating incidents) and build answer packets. The second file, `packets.py`, is a helper which specifies the architecture of the different MQTT packets.

Therefore, the MQTT module is really basic. It simply acknowledges the messages when needed but does not compute anything nor fake any behaviour of an actual MQTT broker. A basic operation allows to figure out that the module is nothing but a fake interface: if someone subscribes to a topic and then publishes to this topic it would never receive the message, as the subscription feature would require to.

This is typical low-interaction honeypot behavior, the aim of the module being to collect data about connections to the fake system. One of the purposes of this thesis is to make it more of a medium-interaction module and to analyze what hackers actually try to do when they get access to a MQTT broker. To achieve that goal, the honeypot needs to be more realistic and to behave like an actual broker. Here are the modifications made to the module, based on MQTT Version 3.1.1 Oasis Standard [5], in order to make it more real.

Important note, the big majority of the modifications are applied to the MQTT module in order to guarantee security and avoid to mess with other parts of dionaea. Modified files are `mqtt.py` and `packets.py` and a file `broker.py` has been added in order to process all the internal computation.

4.2 New MQTT module

4.2.1 Packets

The packet classes of `packets.py` are used to both encapsulate incoming messages in a readable format and as a structure to build response messages. The original MQTT module didn't

support 4 of the packet types informed by the MQTT v3.1.1 Oasis Standard: `PUBREC`, `PUBCOMP`, `UNSUBSCRIBE` and `UNSUBACK`.

`PUBREC` and `PUBCOMP` are used for the QoS level 2 (more info in 3.1.1). Nevertheless, dionaea managed to respond `PUBREC` and `PUBCOMP` messages by altering a `PUBACK` packet's fixed header. Thus, acknowledgment (but no actual service or any computation) for QoS 2 was guaranteed.

In the other hand, the unsubscription mechanism was not supported at all. When a `UNSUBSCRIBE` message arrived it was simply discarded and no `UNSUBACK` packet was sent in response.

The extended MQTT module we implemented now supports all of these 4 packet types (therefore supports all packet types of MQTT v3.1.1). `PUBREC` and `PUBCOMP` packets are separated from `PUBACK`, even if the trick of the original module was not problematic for sending those messages it is not convenient for receiving them (which was not suppose to happen in the low-interaction fashion but is now). The unsubscription service is now available, it helps to make the module more realistic.

4.2.2 Connection establishment

Before the extension, the module took only care of initiating a TCP session with the remote host, and sending back a `CONNACK` along with a status code. Nonetheless, they are many attributes that can be specified during MQTT connection establishment that were not taken into account initially.

Information specified during connection establishment are used to maintain a `Session` object for a specific client. This `Session` will be maintained in the module as long as we know the client is connected and it includes attributes `client_id`, `clean_session`, `subscriptions`, `undelivered_messages`, `is_connected`, `hostname` and `port`. A specific `Session` object can be retrieved at any point in the module as they are registered in a dictionary with `client_id` attributes used as keys of this dictionary.

There was first a bug in the constitution of `CONNECT` packets as the `ProtocolName` field was set on six bytes while the Oasis Standard specifies its length to be four bytes long. This problem impacted all the fields found in the `Connect Flags` as every value was shifted two bytes to the right, making the `cleanSession` or `keepAlive` attributes (Subsection 3.1.2) unreadable. We spotted that bug and corrected it.

There is now a distinction between persistent sessions (`clean_session` flag set to `FALSE`) and non persistent ones (`clean_session` flag set to `TRUE`). In the case of a non persistent session, session state is still maintained for the client as this is necessary in order to use other MQTT functionalities in the module but it is deleted as soon as the client disconnects as explained later in Subsection 4.2.7. In the case of a persistent session, we distinguish several sub-cases like the ones where the client never established a session before from the one where the client already has this `client_id` in a saved session and thus needs to resume it through the one where a take-over needs to be performed because the client got unexpectedly disconnected (not mentioned by the module) and needs to also resume its session. When a client with a persistent session is offline, the messages meant to him are queued until its reconnection.

Error return code `0x02` (identifier rejected) is also supported to close a connection when a client tries to connect using a `client_id` that is already being used by another connected client.

Nonetheless, there is one functionality that we didn't implement here, the LWT one. The reason why is because of the fact that it is used to notify other clients about an ungraceful disconnection. This kind of disconnection can be noticed in the lower layers of dionaea (in the establishment of the TCP connection in modules written in `C`) but as said earlier, we mainly applied modifications to the MQTT module in order to not mess with other parts of dionaea and risking to break it. Another way to notice such type of disconnection would be to used the keep alive specified by clients. However, as the MQTT module is considered as a callback and is only triggered when a MQTT packet is received, the management of periodic tasks like the one of checking if each client is still alive is a bit difficult.

The feature concerning the authentication using a username and a password is not taken into account as it is used to restrict access to a broker. This has no sense for our case since the goal is to have people interacting with it while they are initially not supposed to. Clients (and potential attackers) wishing to use this functionality can nonetheless specify a pair of username/password but this pair will simply be ignored in the processing of a `CONNECT` packet.

```
[16082018 14:02:57] mqtt dionaea/mqtt/mqtt.py:86-info: ---> New message received: CONNECT
[16082018 14:02:57] mqtt dionaea/mqtt/broker.py:44-info: Client ID: ID_160245009
[16082018 14:02:57] mqtt dionaea/mqtt/broker.py:45-info: Clean session: 0
[16082018 14:02:57] mqtt dionaea/mqtt/broker.py:46-info: Username:
[16082018 14:02:57] mqtt dionaea/mqtt/broker.py:47-info: Password:
[16082018 14:02:57] mqtt dionaea/mqtt/broker.py:79-info: ---> Client never established a session before.
[16082018 14:02:57] mqtt dionaea/mqtt/broker.py:309-info: ---> Created a new session for client: ID_160245009
```

Figure 4.1: Connection to the broker.

4.2.3 Subscription to a topic

If the original module was only able to log and acknowledge incoming subscriptions, the extended one can now actually register the subscriptions. The `subscribe callback` from `broker.py` memorizes the subscription of the clients so that if the attacker tries to publish to the topic he just subscribed to, he will receive a proper answer. Each subscription has an associated QoS which corresponds to the higher QoS the client accepts. As described in the Oasis Standard, if a client subscribes to a topic he already subscribed to before, the subscription is updated (with the appropriate QoS) and all the pending messages (not yet acknowledged) are sent again. Once again, this whole mechanism reinforces the impression of reality of the honeypot. The topic wild cards mechanism has also been implemented.

Furthermore, we spotted an error regarding the acknowledgment of `SUBSCRIBE` packets. Those packets are supposed to always be acknowledged by a `SUBACK` packet but the original module sends such a response packet only in case of a `SUBSCRIBE` packet with QoS 1 or 2. The confusion comes from the fact that the specified QoS in the `SUBSCRIBE` message corresponds to the higher QoS supported by the client for the reception of PUBLISH messages, thus the QoS has nothing to do with the acknowledgement of the `SUBSCRIBE` message itself, which should absolutely always happen.

However, one feature is not supported by our implementation (nor by the original one). A `SUBSCRIBE` packet may contain a list of topics to subscribe to rather than only one, but the way the packets are handled by dionaea cuts this list after the first element so that in the MQTT module we only have access to the very first topic (and associated QoS) of the list. One of our ground rule being to only modify the MQTT module, we did not dig deeper in the structure of the honeypot to solve this issue.

```
[16082018 14:04:11] mqtt dionaea/mqtt/mqtt.py:168-info: ---> New message received: SUBSCRIBE QoS 1
[16082018 14:04:11] mqtt dionaea/mqtt/broker.py:155-info: Topic: mqtt/test
[16082018 14:04:11] mqtt dionaea/mqtt/broker.py:156-info: Granted QoS: 1
[16082018 14:04:11] mqtt dionaea/mqtt/broker.py:157-info: Packet ID: 1
[16082018 14:04:11] mqtt dionaea/mqtt/broker.py:206-info: ---> Topic: mqtt/test is valid.
[16082018 14:04:11] mqtt dionaea/mqtt/broker.py:232-info: ---> Added subscription: mqtt/test for client ID_160245009
```

Figure 4.2: Subscription to a topic.

4.2.4 Unsubscription from a topic

`UNSUBSCRIBE` packets were not supported at all as said earlier. Now when a client sends this type of packet, the unsubscription is handled by making sure that the specified topic the client wants to unsubscribe from is a valid topic. The corresponding `UNSUBACK` is also generated and sent.

4.2.5 Publication to a topic

As for the subscription, dionaea is initially only able to log and acknowledge incoming `PUBLISH` messages. We implemented the full publishing feature, relaying publications to subscribers in respect of the desired QoS (for detailed information about QoS management see 3.1.1).

The Oasis Standard is totally respected. The `DUP` flag in `PUBLISH` packet is set to 1 in case of a re-delivery. The `RETAIN` mechanism is also implemented so that when a `PUBLISH` message is received with the `RETAIN` bit set to 1 it is memorized in order to be sent to any new subscriber.

Details of the transmission of `PUBLISH` packets are explained regarding QoS levels in the following section.

```
[16082018 14:08:48] mqtt dionaea/mqtt/mqtt.py:146-info: ---> New message received: PUBLISH QoS 0
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:128-info: Topic: mqtt/test
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:129-info: Message: Test
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:130-info: Retained: True
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:131-info: QoS: 0
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:132-info: Header: 49
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:217-info: ---> Retained message added for topic mqtt/test: (<MQTT
_Publish  HeaderFlags=49 MessageLength=15 TopicLength=9 Topic=b'mqtt/test' Message=b'Test' |>, 0)
[16082018 14:08:48] mqtt dionaea/mqtt/broker.py:256-info: ---> Packet sent :<MQTT_Publish  HeaderFlags=49 Message
Length=15 TopicLength=9 Topic=b'mqtt/test' Message=b'Test' |> to client: ID_160245009
```

Figure 4.3: Publication to a topic.

4.2.6 Quality of Service

Each `Session` object maintains a queue called `undelivered_messages` filled with non acknowledged packets.

Even if unacknowledged messages may be resent later, the Oasis Standard specifies that it is mandatory to resend them only in case of reconnection of a client which had a previous persistent session.
Quote from the MQTT v3.1.1 Oasis Standard section 4.4 Message delivery retry:

" When a Client reconnects with CleanSession set to 0, both the Client and Server MUST re-send any unacknowledged PUBLISH Packets (where QoS > 0) and PUBREL Packets using their original Packet Identifiers [MQTT-4.4.0-1]. This is the only circumstance where a Client or Server is REQUIRED to redeliver messages".

Thus, in order to keep the honeypot simple, we chose to store unacknowledged packets only in case of a possible need to resend them i.e. if the `cleanSession` flag is set to 0.
The whole feature is implemented in accordance to the standard.

The QoS downgrade mechanism is also now supported.

4.2.7 Disconnection

When disconnecting a client, a check is now performed in accordance with its connection.

If the client has specified a persistent session at the time he tried to establish a connection with the broker, we need to keep the state created for this client in memory and we also need to specify that the client is now disconnected by setting the attribute `is_connected` to `FALSE`.

In the other case where the client has specified a non persistent session, we need to get rid of the state we created for him along with its association with existing subscriptions.

```
[16082018 14:13:55] mqtt dionaea/mqtt/mqtt.py:183-info: ---> New message received: UNSUBSCRIBE
[16082018 14:13:55] mqtt dionaea/mqtt/broker.py:175-info: Topic: mqtt/test
[16082018 14:13:55] mqtt dionaea/mqtt/broker.py:206-info: ---> Topic: mqtt/test is valid.
[16082018 14:13:55] mqtt dionaea/mqtt/broker.py:243-info: ---> Deleted subscription: mqtt/test for client ID_1602
45009
[16082018 14:13:55] mqtt dionaea/mqtt/mqtt.py:203-info: ---> New message received: DISCONNECT
```

Figure 4.4: Unsubscription from a topic and disconnection from the broker.

4.2.8 Broker information topic

We implemented some of the `$SYS` topics for clients to find information about the broker. Those topics are used by MQTT brokers to share information about the state of the system (e.g. number of clients, topics, etc.). They include some possible topics like the ones giving information about the build time stamp of the broker, its version, the number of clients connected, the number of messages sent, etc.

4.2.9 Logging to the database

Support for newly considered attributes like QoS or `RETAINED` flag has been added in the SQL database and can be observed in retrospect. The corresponding incidents have also been added.

4.2.10 Summary

Features	Old Module	New module	Oasis Standard
Connection Establishment	✓	✓	✓
Session Management	✗	✓	✓
Persistent Session	✗	✓	✓
Offline Message Queuing	✗	✓	✓
Keep Alive	(✓)	(✓)	✓
Last Will and Testament	✗	✗	✓
Authentication	✗	✗	(✓)
Subscription Acknowledgment	✓	✓	✓
Actual Subscription	✗	✓	✓
Wild cards	✗	✓	✓
Multiple Subscriptions	✗	✗	✓
Unsubscription Acknowledgment	✗	✓	✓
Actual Unsubscription	✗	✓	✓
Publication Acknowledgment	✓	✓	✓
Actual Publication	✗	✓	✓
`DUP` Flag	✗	✓	✓
`RETAIN` Flag	✗	✓	✓
QoS 0	✓	✓	✓
QoS 1 Acknowledgment	✓	✓	✓
QoS 1 Redelivery	✗	✓	✓
QoS 2 Acknowledgment	(✓)	✓	✓
QoS 2 Redelivery	✗	✓	✓
QoS Downgrade	✗	✓	✓
Disconnection	✓	✓	✓
`$SYS` topics	✗	(✓)	✓

Chapter 5

Evaluation

In this chapter, we first describe the different tools that we used to conduct the extension of the honeypot. Then, we assess our extension of the MQTT module thanks to a conformity test suite and some load tests on the honeypot. Finally, by the means of a scenario, we show how easy a basic attack can be performed against a MQTT broker.

5.1 Tools

We used several tools throughout development and testing phase in order to evaluate our implementation, they are described in this section.

5.1.1 Oracle VM VirtualBox

It is absolutely necessary to use a Virtual Machine (VM) while working with honeypots in case an attacker succeeds to get out of it and gets access to the system. We chose VirtualBox[1], developed by Oracle, it is one of the most famous virtualization software. VirtualBox has the advantages to be open-source, widespread and easy to use. Thanks to this software we virtualized an Ubuntu 14 machine which is a favorable environment to run dionaea.

5.1.2 Wireshark

Wireshark[2] is the world's widely-used network protocol analyzer. It is the one most needed software when it comes to analyze traffic. It allowed us to check the validity of packets transmitted by our module, understand the architecture of incoming messages and TCP encapsulation.

5.1.3 Eclipse Paho

This Eclipse project[3] provides open-source client implementations of MQTT in many programming languages, we used it in Python. The library implements a client class that can be used to add MQTT support to a python program as well as some helper functions to make publishing one off messages to an MQTT server very straightforward.

This is the primary tool we used to simulate clients and write scripts to communicate with our MQTT module. Some Paho scripts were also running to simulate activity while we made the honeypot available on a server.

[1] `https://www.virtualbox.org`
[2] `https://www.wireshark.org`
[3] `https://www.eclipse.org/paho/`

5.1.4 mqtt-spy

mqtt-spy[4] is an open source utility intended to help people with monitoring activity on MQTT topics. This software, built upon Paho, is simple to use thanks to a Graphical User Interface (GUI). We used it for its simplicity to quickly send a message to the broker when needed. Plus, working with several client software (Paho, spy, MQTTBox) guarantees the quality of our implementation.

5.1.5 MQTTBox

MQTTBox[5] is an app comparable to mqtt-spy for its client component and its GUI. As said above, we used it for the variety of client implementation, but also for its load test feature. Indeed, MQTTBox provides a component able to flood a broker with subscription or publication by launching up to 10 instances working simultaneously, which was useful for stress testing.

5.2 Testing

5.2.1 Conformity tests

In order to test the behaviour of the MQTT module, we used a test suite highly inspired by the one provided by the Eclipse Fundation under the Paho project[6].

This suite is made of eleven complete test cases checking all the major features of MQTT v3.1.1. We performed seven out of those eleven tests, we skipped the ones related to the not implemented features LWT and keep alive, as well as publish to dollar topics and the subscribe failure. The latter consists of testing to subscribe to a topic which is not allowed to be subscribed to, which gives no result and makes no sense in our case since we leave the subscription to any topic opened on purpose. In the other hand, the "publish to dollar topics" test checks that it is forbidden to publish to a topic which starts with a $ sign. It is specify in this test suite that this test may fail since this feature is not mandatory. Indeed, we did not restrict the publication to $ topics in order to give more freedom to potential attackers.

The seven other test cases are described bellow. Note that all these tests are binary, it is a pass or fail. Our implementation succeeds in all of these.

Basic Test

This test checks the general behavior of the MQTT broker by trying to connect, subscribe and publish with all QoS levels. It does so through these steps :

1. A client connects and disconnects.
2. The client reconnects, the acknowledgement is checked.
3. The client subscribes to several topics.
4. The client publishes on those topics, one different publish per QoS.
5. The script checks that publish messages have been transmitted and acknowledged properly.
6. Try to connect a new client with a wrong protocol name in the header and expect a connection refuse.

[4] `https://github.com/eclipse/paho.mqtt-spy/wiki`
[5] `http://workswithweb.com/mqttbox.html`
[6] `https://github.com/eclipse/paho.mqtt.testing`

Retained Messages

The retain feature is tested by pushing retained messages as well as clearing them.

1. A client connects.
2. The client publishes on different topics with the three QoS levels and the `RETAIN` flag set.
3. The client subscribes to those topics, by using wild cards by the way.
4. The script checks that retained messages are received.
5. The client disconnects and reconnects.
6. The client publishes empty messages on the topics (same as previously) with the three QoS levels and the `RETAIN` flag set. The combination empty message and flag set means that the retained message should be withdrawn.
7. The client subscribes to those topics.
8. The script checks that no retained message is received anymore.

Zero Length Client ID

This test case checks that connection attempts with empty `client_id` field are rejected if a persistent session is required and that it is accepted and a random `client_id` is set by the broker in case of a `clean_session = True` flag.

1. A client with no `client_id` attempts to connect with `clean_session = False` flag.
2. The script checks that this is rejected.
3. A client with no `client_id` attempts to connect with `clean_session = True` flag.
4. The script checks that this is accepted.

Offline Message Queuing

The purpose of this test is to verify that the messages intended for a disconnected client with a persistent session are queued until its reconnection.

1. A client A connects with `clean_session = False` and subscribes to different topics.
2. The client A disconnects.
3. Another client B connects (with non persistent session) and publishes on different topics with the three QoS levels before disconnecting.
4. The client A reconnects.
5. The script checks that the `CONNACK` packet has the Session Present flag set.
6. The script checks that QoS 1 and 2 messages have been received and notify if QoS 0 messages have also been (which is not mandatory).

Overlapping Subscriptions

It may happen that there is more than one matching subscription per client for a topic (due to wild cards for instance). In such a case the broker may send back a single message with the highest QoS of any matching subscription, or send one message for each subscription with a matching QoS. Note that in our case, the second option has been chosen for practical reasons.

1. A client connects.
2. The client subscribes to different topics which overlap.
3. The client publishes on such a topic with QoS 2.
4. The script checks that one of the two admissible responses is received.

Redelivery on Reconnect

When a QoS 1 or 2 exchange has not been completed, the server should retry to send the appropriate packets. This test verifies this behavior.

1. A client A connects with `clean_session = False` and subscribes to different topics.
2. The client pauses i.e. stop responding to incoming publishes.
3. The client publishes on the topics with QoS 1 and 2.
4. The client disconnects.
5. The script checks that no acknowledgment has been received.
6. The client resumes and reconnects.
7. The script checks that the acknowledgments have been received.

Unsubscribe

This test checks that the unsubscription mechanism is properly implemented.

1. A client A connects and subscribes to three different topics.
2. The client A unsubscribes from one of these topics.
3. A client B connects and publishes to the three topics with QoS 1.
4. The script checks that the client A has received only two out of the three messages.

5.2.2 Load tests

Publish/Subscribe

The application *MQTTBox*[7] embeds two load test features for publication and subscription. It allows to set up to 10 instances which flood the broker with publish or subscribe packets simultaneously. We used QoS 2 for those tests in order to maximize their efficiency.

Unfortunately, the subscribe feature is broken since for mysterious reason it sends `UNSUBSCRIBE` packets instead of `SUBSCRIBE`. Nevertheless, we would expect our implementation to react similarly to subscribe flood than to publish in QoS 1 since it involves nearly the same kind of packets.

[7] `http://workswithweb.com/mqttbox.html`

Few instances can never disturb the broker, packets are sent at the maximum rate the application can achieve (~150/sec) and are acknowledged right away by the honeypot. Also this does not prevent other (regular) clients to communicate with the broker properly. On the other hand, with 8 to 10 instances we notice packet and most importantly connection drops. The flood test never succeeds and other regular communications in QoS 1 or 2 are highly slowed or even cut. QoS 0 still works fine.

However, once the flood ends, the broker recovers and is again accessible to regular clients.

Connect flood

We proceeded to the same kind of flood test than above but this time with connections. The test consist of a paho script sending `CONNECT` requests from different ports with different random `client_id` at maximum rate (again ~150/sec). All packets are acknowledged immediately and the broker is not disturbed at any time, connections from other regular clients are still possible.

SYN flood

The last test performed is a TCP `SYN` flood attack. About 50.000 TCP `SYN` packets are sent each second to the port 1883 (used by the honeypot) thanks to the command `hping`. This attack makes the broker mostly unavailable, connections are really slow and publications in QoS 0 sometimes make it to the subscribers. QoS 1 and 2 are completely unusable.

Once again, when the attack stops the broker is normally usable again.

5.3 Results

We let the honeypot running on a server for a total period of approximately one month using three different versions as we brought new functionalities into the MQTT module each time. However, although we did test each functionality locally in several ways before integrating them within the exposed honeypot, we have been confronted with bugs that sometimes didn't occurred in local, making them more difficult to fix.

5.3.1 MQTT related results

From the 540430 connection attempts with the honeypot, 14 were related to MQTT, which is a small amount. Also, clients that interacted with the honeypot didn't have an abnormal behaviour: they basically established a connection with the broker, subscribed to topics for some of them and waited for incoming messages on these topics.

However, we can still extract some interesting knowledge from the results that were captured in the dionaea SQLite database and that are discussed regarding source IP addresses, and MQTT subscriptions.

Source IP addresses

We observed 8 unique IP addresses from the results (Figures 5.1, 5.2 and 5.3). While some of them originate from a specific individual most of the time not hidden behind a proxy, others seem to come from servers owned by Shodan[8] that are used to scan devices and to index them as vulnerable devices on their website.

Shodan scanner can find systems connected to the Internet like traffic security cameras, home heating systems, water plants, power grids etc. with many of these having quite a lot of

[8] `https://www.shodan.io/`

vulnerabilities [48]. Moreover, it can identify few information about these systems from their physical location to the type of software they are running passing through the Autonomous System Number (ASN) in which they are located.

This makes the MQTT broker more accessible to attackers as Shodan is extensively used for this purpose. It is often dubbed as "Google for hackers". An example of IP lookup result for this kind of server is shown in Figure 5.4.

connection	connection_protocol	local_host	local_port	remote_host	remote_port
1310	mqttd	[redacted]	1883	198.20.69.98	48515
13727	mqttd	[redacted]	1883	125.64.94.20	47250
13729	mqttd	[redacted]	1883	125.64.94.20	52462
15671	mqttd	[redacted]	1883	71.6.146.185	52873

Figure 5.1: Connections information collected in the first version.

connection	connection_protocol	local_host	local_port	remote_host	remote_port
60517	mqttd	[redacted]	1883	71.6.165.200	39130

Figure 5.2: Connections information collected in the second version.

connection	connection_protocol	local_host	local_port	remote_host	remote_port
163278	mqttd	[redacted]	1883	118.193.31.181	52786
163279	mqttd	[redacted]	1883	118.193.31.181	41858
163284	mqttd	[redacted]	1883	118.193.31.181	46838
196446	mqttd	[redacted]	1883	185.35.63.245	34471
300763	mqttd	[redacted]	1883	198.20.70.114	43157
307846	mqttd	[redacted]	1883	80.82.77.139	58841
415427	mqttd	[redacted]	1883	118.193.31.181	35356
415428	mqttd	[redacted]	1883	118.193.31.181	52824
415430	mqttd	[redacted]	1883	118.193.31.181	57800

Figure 5.3: Connections information collected in the third version.

Subscriptions

Not all clients that connected themselves to the honeypot made a subscription. For those who did, it only concerns two well known topics, `#` and `$SYS/#` as shown on Figures 5.5, 5.6 and 5.7. Both individuals and Shodan scanning servers appear in the subscriptions.

Logically, Shodan servers subscribe to the `$SYS/#` topic in order to get specific information related to the broker as explained in Subsection 4.2.8. This way, it is able to list the set of topics currently registered in the broker memory.

Concerning the individuals, they may be aiming to extract confidential information that is exchanged between IoT devices in a network. This can have a limited impact for a MQTT broker that only processes messages about temperature in a room, e.g, but can be way more dangerous when more confidential data, like the location of a specific device, is exchanged.

5.3.2 General results

Following results only come from the third version of the honeypot as we had some problems merging the different databases into a single one. This version is the one that have run the most and that gave the more results.

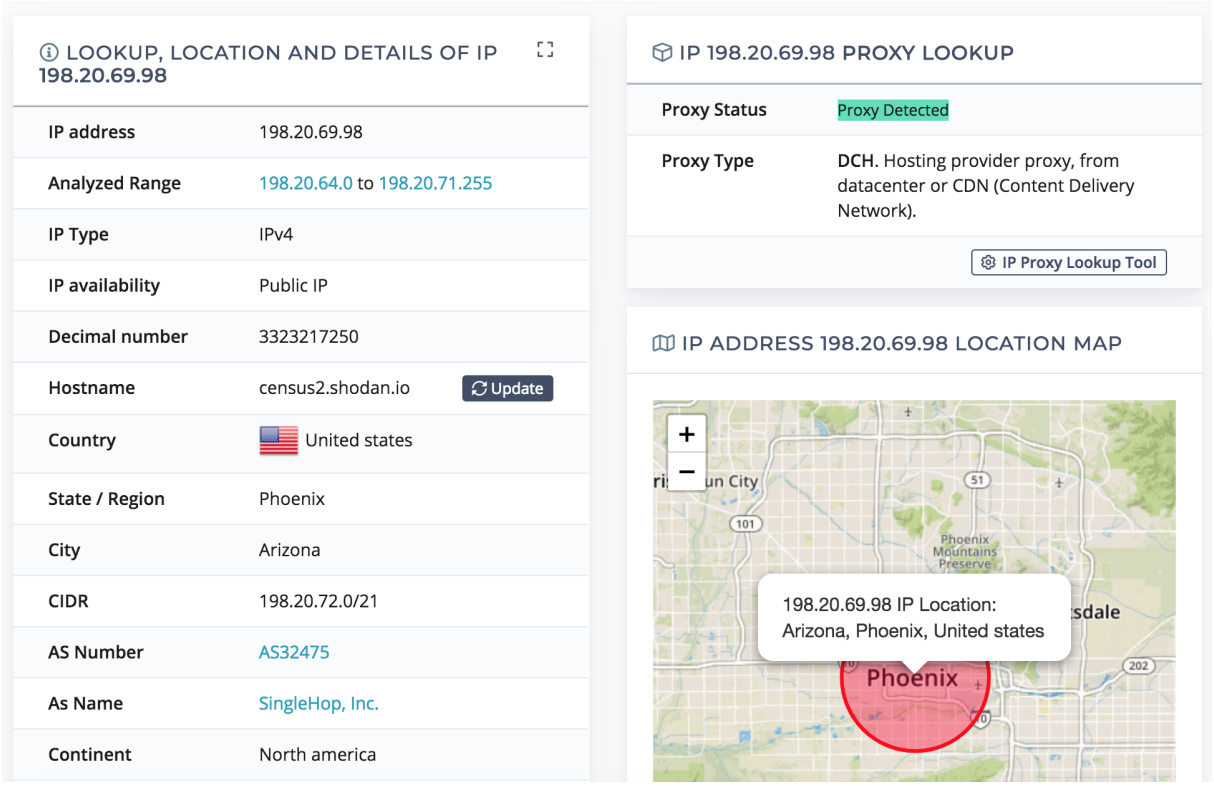

Figure 5.4: Result of an IP lookup for `198.20.69.98`

mqtt_subscribe_command	connection	mqtt_subscribe_command_messageid	mqtt_subscribe_command_topic
1	1310	1	#
2	13729	2	$SYS/#
3	15671	1	$SYS/#

Figure 5.5: Subscriptions information collected in the first version.

mqtt_subscribe_command	connection	mqtt_subscribe_command_messageid	mqtt_subscribe_command_topic	mqtt_subscribe_command_qos
2	60517	1	$SYS/#	0
3	60517	2	#	0

Figure 5.6: Subscriptions information collected in the second version.

mqtt_subscribe_command	connection	mqtt_subscribe_command_messageid	mqtt_subscribe_command_topic	mqtt_subscribe_command_qos
2	300763	1	$SYS/#	0
3	300763	2	#	0
4	307846	1	$SYS/#	0

Figure 5.7: Subscriptions information collected in the third version.

Targeted ports

Besides MQTT, other more famous protocols were targeted as shown in Figure 5.8. In these ones, we find the well-known ports 80/443/8080 for `HTTP`, 22 for `SSH` and 23/2323 for `Telnet`.

1. `Telnet` is obviously the most targeted port with 97% of the top 10 count. As explained earlier, malicious people trying to communicate on this port aim to find an open door inside the concerned device by using weak credentials like the Mirai malware does.

2. Then, we have `SSH` port. This port can be targeted by brute force attacks, like in `Telnet`, concerning devices that have weak credentials.

local_port	count
23	376284
22	1193
81	1178
80	1134
445	1050
5555	830
8080	688
443	638
3389	606
2323	476

Figure 5.8: Most targeted ports in the third version.

3. Ports used by the `HTTP` protocol are the ones concerning web servers communications. They can be targeted by DoS attacks and more specifically by `HTTP` flood.

4. Port 445 is a traditional Microsoft networking port used by the `Server Message Block (SMB)` protocol. It is known to be targeted by the WannaCry ransomware that did a lot of damage back in 2017 by infecting Windows computers with an encryption of all the files that could be found on the machine, making them impossible for users to access without paying a ransom [19].

5. Finally, we have port 3389 used by `Remote Desktop Protocol (RDP)`, a proprietary protocol developed by Microsoft. It provides a graphical access to a client from a server by using credentials. Poorly secured `RDP` devices can give attackers a potential entry point into enterprise networks that use this technology [52].

Source IP addresses

We decided to take a look at source IP addresses that targeted the most famous ports discussed just before (`Telnet`, `SSH` and `HTTP`) to see if we could extract some knowledge about them. Obviously, the most appearing IP addresses targeted the `Telnet` protocol. Nonetheless, we didn't notice any particular information when doing an IP lookup for theses addresses.

remote_host	count
178.128.39.135	24255
185.244.25.219	19275
159.65.179.186	18828
206.189.209.11	17983
80.211.76.19	16740
195.43.95.179	15671
209.97.138.248	12281
80.211.23.64	11883
206.189.168.19	9898
159.65.232.123	9497

Figure 5.9: IP addresses that come up most often in the third version.

5.4 Attack simulation

We will here demonstrate with an attack scenario how easy it is to conduct a basic attack against a MQTT broker that provides no security mechanism. The broker in question will be ours which is running as a module of the dionaea honeypot.

5.4.1 Find an open MQTT broker

Finding an open broker is very easy thanks to Shodan. When entering "MQTT" in the search bar, a lot of brokers come up. To restraint the search, we can specify a `Connection Code` of `0`: this states that the broker does not require any authentication from clients. The results returned by Shodan indicate that 19,488 MQTT brokers satisfy the condition.

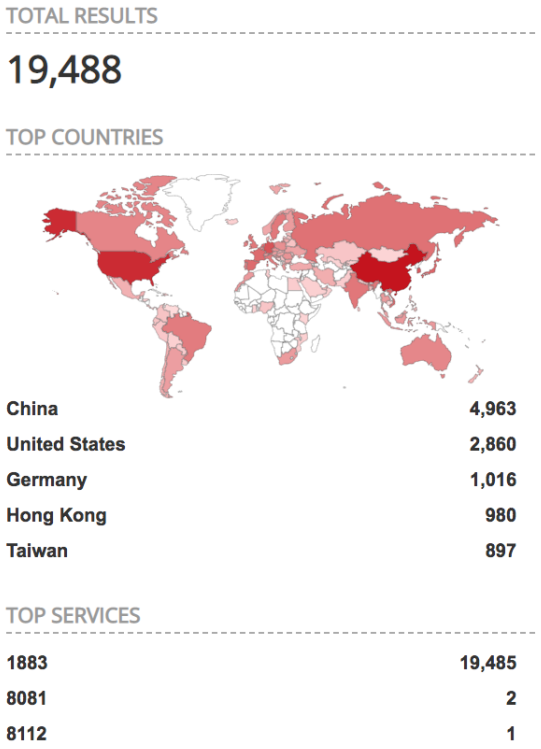

Figure 5.10: Results returned by Shodan concerning open MQTT brokers.

This is the graphical way to do it but it is also possible, thanks to Shodan API, to automate this process using a script allowing to search for specific devices and to conduct actions (as the ones that follow) for each one found.

The broker targeted is the one having the IP address of the server running the dionaea honeypot with our extension. The victim is chosen, we now need to establish a connection.

5.4.2 Establish a connection using a MQTT client

There are quite a few MQTT clients available online. The one we will use is mqtt-spy[9] which is a `JavaFX` application providing a neat GUI.

We specify the required information to establish the connection with the broker by inputting its address, a client ID and a clean session flag set to `TRUE`.

5.4.3 Obtain broker related information

Once the connection is established with the broker, we can try to pull some information related to the broker. To do so, we subscribe to topic `$SYS/#` and wait for incoming messages.

[9] `https://kamilfb.github.io/mqtt-spy/`

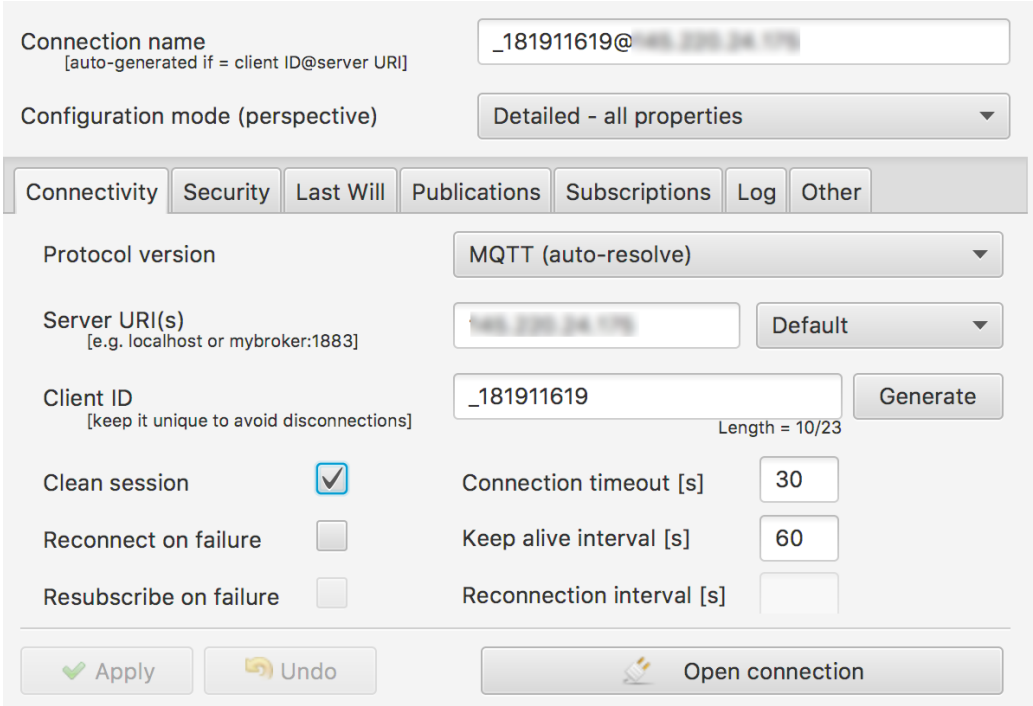

Figure 5.11: Connection parameters in mqtt-spy to connect to 145.220.24.175.

As soon as the subscription to the topic is done, messages get to our client. We found out that messages published on topic `$SYS/broker/timestamp` and `$SYS/broker/version` are retained messages directly sent on subscription to clients and give information about the build time of the broker and its version (Figure 5.12).

5.4.4 Observe exchanged messages through the broker

We now decide to subscribe to the topic `#` in order to receive all the messages that are published on any topic in order to extract interesting information.

After a while, some messages get to our malicious client (Figure 5.13). From the messages received on the specific topic, we can infer that this broker is used for personal use in home automation. We identified a pattern used in the messages and we are now able to send our own illegitimate ones to operate on the smart devices used.

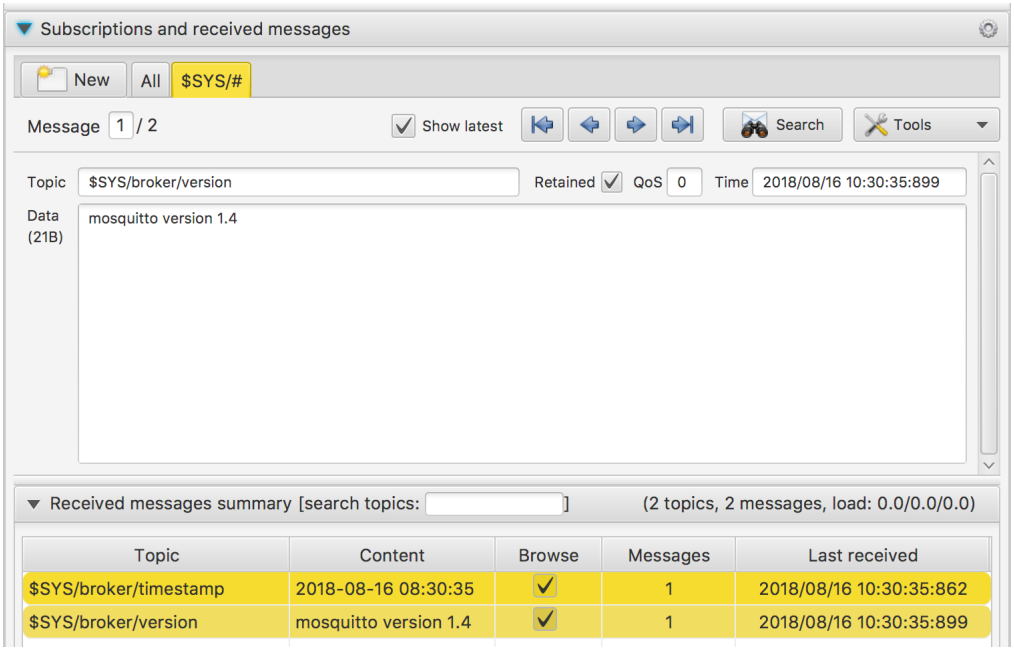

Figure 5.12: Messages received after subscribing to `$SYS/#` topic.

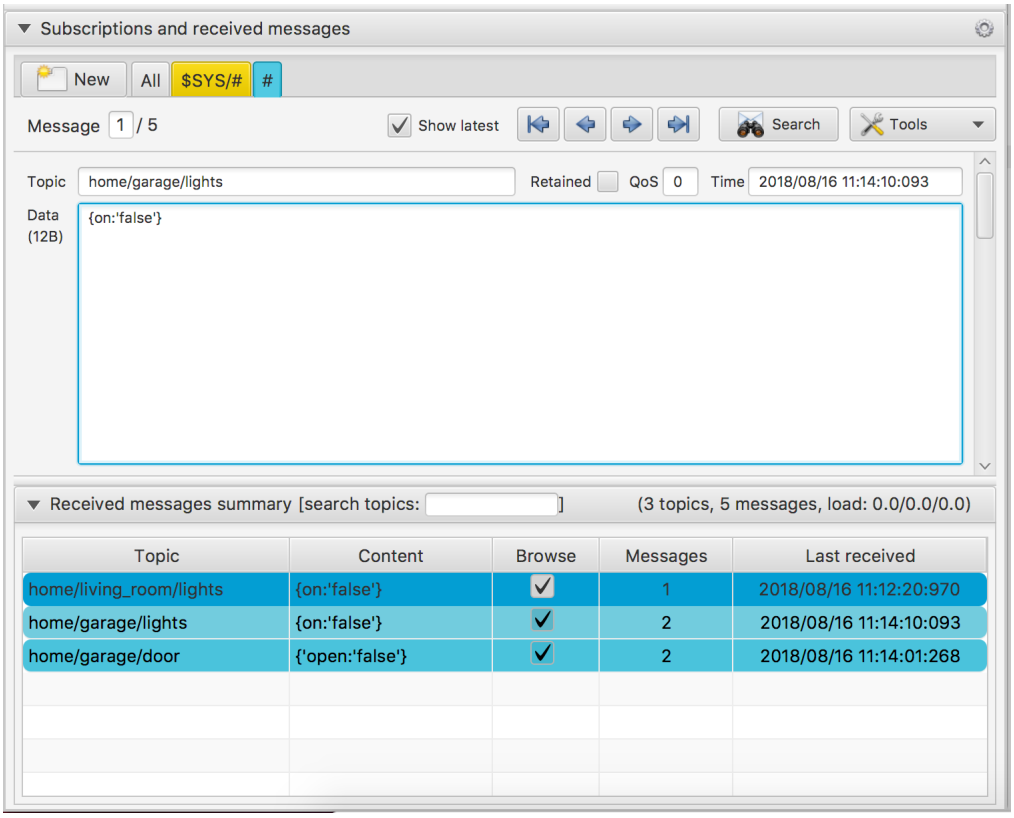

Figure 5.13: Messages received after subscribing to `#` topic.

Chapter 6

Recommendations

IoT presents vulnerabilities at every layer of its architecture. The fact that it is a recent trend makes manufacturers and developers overlook the security of the products and technologies they are making, with productivity being the first point of their checklist as security functionalities would obviously increase their cost and development time. Business of other people using these technologies is affected when security vulnerabilities are spotted and exploited. A defence-in-depth strategy is required where an active participation of all the concerned players is also needed.

Much of the potential harm that can be done using IoT devices is avoidable by enforcing a few rules. Here, we will present some of them described by George Corser and Jared Bielby [13] in their IEEE white paper about IoT security best practices, in accordance with the vulnerabilities that were discussed throughout the thesis.

Make hardware tamper resistant Many IoT devices are designed to be unattended. As we cannot always monitor them, making these devices tamper-proof in addition to keep them physically isolated can help preventing attackers from reaching data and from altering the devices for impersonation attacks. Physical security of endpoints could include small simple plastic devices, port locks or camera covers and approaches to automatically disable the device when it is tampered with.

This kind of rule could be applied to devices used by protocol like LoRa, for example, where a malicious MCU attack is possible by replacing the legitimate MCU to send malicious data on behalf of the node.

Provide for firmware updates/patches As vulnerabilities are often discovered once the devices are available for sale, there is a need for manufacturers and developers to provide a way for the customers of upgrading the devices firmware or to correct bugs thanks to patches. In addition, customers need to be informed about the time span during which the device is supported and what actions should be taken once this period is due.

Cryptographic key compromising problems are often subject to vulnerabilities. Protocols using weak key exchange mode may be required to be improved and automatically updated such that customers no longer use a deprecated version including vulnerabilities that are known from the attackers.

Use strong authentication Default credentials (e.g., username is "*admin*" and password is "*admin*") are as dangerous as common credentials (e.g., username is "*root*" and password is

"*12345*") and thus should not be used. A lot of tools on the Internet used to do brute force authentication for some known protocols can easily be found. Also, devices should not include debugging mode with the use of secret credentials established by the device's developer as they can be used as backdoors.

On top of secure credentials, two-factor authentication is recommended. This process requires a user to verify his identity by passing another authentication system, inputting a possession factor (something the user has) or an inherence factor (something the user is).

Weak authentication systems in IoT devices is the scourge that allowed IoT botnets to grow to such an extent. By having unique secure credentials to log in into every IoT devices, some botnets like the Mirai one would have been nonexistent in the best case.

Use strong encryption and secure protocols Even if device passwords are secure, there is still a need to secure the communications between them. IoT includes a bunch of protocols for every kind of IoT solution needed. As the majority of them are resource constrained, IoT devices are not always able to implement advanced security mechanisms. Nonetheless, they should use the strongest possible one when it is possible.

We have seen efforts from organizations like the National Institute of Standards and Technology (NIST) to develop lightweight encryption standards for resource-constrained devices [10] like the ones we can find in IoT but there is still a need to get the public involved in order to contribute to this type of initiative.

6LoWPAN extension for Internet Protocol Security (IPsec) [47] is a type of security mechanism that can be used by 6LoWPAN sensor nodes to communicate in a secure way with the Internet. It provides an end-to-end secure communication while being able to use IPv6 mechanisms.

Some protocols provide built-in options to secure data transfers like the MQTT protocol by using Transport Layer Security (TLS). When such option is available, it should be used to avoid any malicious entity from being able to read exchanged messages or even authenticate itself as a legitimate side in a communication.

Minimize device bandwidth The principle of least privilege is the practice of giving the bare minimum of resource, computation capabilities to a user, device, program etc. they need to perform their work. This practice should be used for IoT devices such that, if they are compromised, the effects of malicious actions they carry out have limited impacts.

This makes even more sense as the number of IoT devices is intented to explode in the future. Botnets recruiting IoT devices would cause less damage if their zombies had limited bandwidth resources or a way to downgrade these when they are compromised.

Protect sensitive information IoT products should only collect necessary data for its proper functioning. Also, users of these products should be informed of what is collected about them and how this information is being used with the aim of reducing the risk of data leakage and to protect consumers privacy.

As discussed earlier, IoT solutions like frameworks that allow third-party to develop their own applications tend to collect more data than what is really needed. In addition, they do not provide any way for the user to control how the sensitive data inputted should be used inside applications, benefiting data leakage as dangerous flows are not forbidden.

Chapter 7

Conclusion and future work

The first aim of this thesis was to study the possible points of attack in commonly used IoT technologies and define a list of IoT attacks that can occur according to the mentioned points of attack. We did so by making a survey of common IoT technologies used and presenting some vulnerabilities they have and that are exposed in papers. Attacks that we mentioned according to the possible points of attack can be grouped in different types: DoS, cryptographic key compromising, violation of privacy, impersonation and the most known one, botnets. This part allowed to brought to light the multiple security issues that can be found in IoT devices, protocols and networks and also the fact that this field still needs more involvement in a controlled security from manufacturers, developers and customers.

The second aim consisted in clearly understanding how IoT attacks are performed in practice and how to defend against them thanks to an extended honeypot for IoT. As the MQTT protocol seemed to gain more and more popularity and seemed to be attacked in different ways, we decided to base the extension of the honeypot on this protocol. This extension was possible thanks to the dionaea honeypot that already proposed a very basic module for the protocol. The new MQTT module was tested in order to ensure a proper functioning. The few results obtained by the honeypot showed that individuals as well as information inventory servers like Shodan's interact with the module inside the honeypot certainly to extract information exchanged between the clients of the MQTT protocol.

Concerning the limitations of our work, they are mainly related to the results obtained by the honeypot. In fact, from the 540430 connection attempts with the honeypot, only 14 were related to MQTT. The reason of this few results is not known from a reliable source but we can emit two possible hypotheses:

1. The visibility of the MQTT module running and the one of the honeypot itself is not good.
2. The attackers' interest for this kind of protocol is still not important enough.

Regarding future work, the first step would be to migrate from dionaea 0.7.0 to 0.8.0, indeed the latest version of the honeypot was launched while we were working on this thesis. Information about the migration are available in the official documentation[1]. Once dionaea up-to-date, the next step would be to submit a pull request to the original project to make our work publicly available. Also, dionaea is lacking of documentation and this is a bad habit that we should not reproduce, thus it would be nice to propose a complete documentation for this module while submitting the pull request.

[1] https://dionaea.readthedocs.io/en/latest/migration.html

Moreover, we think that we could get really decent results by running the MQTT honeypot for much longer (few months). MQTT attacks are currently quite rare but it is possible to catch some on the long run.

In terms of features, our module still doesn't fulfill the MQTT v3.1.1 Standard. The implementation of LWT and keepalive would be a great improvement though it requires to get out of the module and dive a bit deeper into dionaea. MQTT 5 could be the object of another module rather than an improvement of this one. However, ours would be a really good base for the implementation of the version 5.

Appendix A

Install and run the honeypot

Scripts have been designed in order to facilitate the deployment and the functioning of the honeypot. The script `deploy_honeypot.sh` takes care of performing several steps:

1. Install required build dependencies before configuring and building dionaea.
2. Run `autoreconf` to build or rebuild the build scripts.
3. Run `configure` to configure the build scripts.
4. Run `make` and `make install` to install the honeypot.
5. Change the ownership of some directories where files have to be written as the honeypot is launched with a switch of Linux user and group.
6. Modify the config file `dionaea.cfg` such that only MQTT is used by the honeypot.
7. At this point the new honeypot can be found in the directory `/opt/dionaea` and it is launched on the server with a `nohup` command to let it run in background.
8. Script `update_sys.py` is finally launched also in background to push broker information related to `$SYS` topics.

```
git clone https://github.com/ysmal/dionaea.git
cd dionaea
sudo ./deployement/deploy_honeypot.sh
```

Appendix B

Run the tests

The `paho.mqtt.testing`[1] project has been added to ours. Thus, running the tests is easy and only requires `python3`.

1. Launch the honeypot.
2. Go to `paho.mqtt.testing`'s interoperability folder.
3. Run the client test with the test of your choice.

```
cd dionaea/paho.mqtt.testing/interoperability
sudo ./deployement/deploy_honeypot.sh
python3 client_test Test.<test_name>
```

Where the test name can be:

- `testBasic`;
- `test_retained_messages`;
- `test_zero_length_clientid`;
- `test_offline_message_queueing`;
- `test_overlapping_subscriptions`;
- `test_redelivery_on_reconnect`;
- `test_subscribe_failure`;
- `test_unsubscribe`.

[1] `https://github.com/eclipse/paho.mqtt.testing`

Glossary

6LoWPAN IPv6 over Low-Power Wireless Personal Area Network. 7, 8, 10, 13, 53

ASN Autonomous System Number. 46

BLE Bluetooth Low Energy. 10, 14

DAG Directed Acyclic Graph. 8

DAO Destination Advertisement Object. 8

DDoS Distributed Denial-of-Service. 20, 21, 31

DIO DODAG Information Object. 8, 9, 13

DIS DODAG Information Solicitation. 8, 13

DODAG Destination Oriented Directed Acyclic Graph. 8, 9, 13

DoS Denial-of-Service. 13, 31, 48, 54

EPC Electronic Product Code. 16

FFD Fully Functional Device. 7, 10

GSoC Google Summer of Code. 32

GUI Graphical User Interface. 42

HTTP Hypertext Transfer Protocol. 11, 20, 21, 48

IETF Internet Engineering Task Force. 30

IoT Internet of Things. 1, 3–8, 10, 11, 13, 14, 17, 19–21, 23, 30–33, 46, 52–54

IP Internet Protocol. 11, 20, 21, 45–49

IPsec Internet Protocol Security. 53

IPv6 Internet Protocol version 6. 8, 13, 53

LLN Low-power and Lossy Network. 8

LoRa Long Range. 11, 19, 52

LoRaWAN Long Range Wide Area Network. 11, 14

LPWAN Low Power Wide Area Network. 11

LR-WPAN Low-Rate Wireless Personal Area Network. 7

LTK Long Term Key. 14

LWT Last Will and Testament. 26, 37, 42, 55

M2M Machine-to-Machine. 11

MAC Medium Access Control. 7, 11, 14

MCU Microcontroller Unit. 19, 52

MQTT Message Queuing Telemetry Transport. 1, 5, 11, 23–26, 28–30, 35–39, 41, 42, 45–49, 53–56

NFC Near Field Communication. 9, 17, 18

NIST National Institute of Standards and Technology. 53

OS Operating System. 32

OTA Over-The-Air. 14, 15

P2P Peer-to-Peer. 21

PAN Personal Area Network. 7, 10

PDoS Permanent Denial-of-Service. 20

QoS Quality of Service. 24–29, 36–40, 42–45

RDP Remote Desktop Protocol. 48

RF Radio Frequency. 10, 11

RFC Requests For Comments. 30

RFD Reduced Functional Device. 7, 10

RFID Radio-Frequency Identification. 9, 13, 15, 16

RPL Routing Protocol for Low-Power and Lossy Networks. 8, 9, 13, 16

SCP Secure Copy Protocol. 32

SFTP SSH File Transfer Protocol. 32

SIPHON Scalable high-Interaction Physical HONeypot. 32

SMB Server Message Block. 48

SQL Structured Query Language. 34, 39

SSH Secure Shell. 20, 21, 32, 33, 47, 48

STK Short Term Key. 14

TC Technical Committee. 28, 29

TCP Transmission Control Protocol. 20, 36, 37

TCP/IP Transmission Control Protocol/Internet Protocol. 25

TK Temporary Key. 14

TLS Transport Layer Security. 53

TTL Time to Live. 29

VM Virtual Machine. 41

ZDO Zigbee Device Object. 10

Bibliography

[1] Aras, E., Small, N., Ramachandran, G. S., Delbruel, S., Joosen, W., and Hughes, D. Selective jamming of lorawan using commodity hardware. *CoRR abs/1712.02141* (2017).

[2] Archer, Q. Rfid: a threat to privacy?, April 2005. https://www.computerweekly.com/opinion/RFID-a-threat-to-privacy.

[3] Baecher, P., Koetter, M., Holz, T., Dornseif, M., and Freiling, F. The nepenthes platform: An efficient approach to collect malware. In *International Workshop on Recent Advances in Intrusion Detection* (2006), Springer, pp. 165–184.

[4] Bandyopadhyay, D., and Sen, J. Internet of things: Applications and challenges in technology and standardization. *Wireless Personal Communications 58*, 1 (May 2011), 49–69.

[5] Banks, A., and Gupta, R. Mqtt version 3.1.1 plus errata 01, Dec 2015. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[6] Bonomi, F., Milito, R., Zhu, J., and Addepalli, S. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing* (New York, NY, USA, 2012), MCC '12, ACM, pp. 13–16.

[7] BOTEZATU, B. Hide and seek iot botnet resurfaces with new tricks, persistence, May 7 2018. https://labs.bitdefender.com/2018/05/hide-and-seek-iot-botnet-resurfaces-with-new-tricks-persistence/.

[8] BOTEZATU, B. New hide 'n seek iot botnet using custom-built peer-to-peer communication spotted in the wild, Jan. 24 2018. https://labs.bitdefender.com/2018/01/new-hide-n-seek-iot-botnet-using-custom-built-peer-to-peer-communication-spotted-in-the-wild/.

[9] Botticelli, B. State of the art: Iot honeypots, Sep 2 2017. https://fr.slideshare.net/BiagioBotticelli/state-of-the-art-iot-honeypots.

[10] Boutin, C. Nist issues first call for 'lightweight cryptography' to protect small electronics, Apr 18 2018. https://www.nist.gov/news-events/news/2018/04/nist-issues-first-call-lightweight-cryptography-protect-small-electronics.

[11] Columbus, L. 2017 roundup of internet of things forecasts, Dec. 10 2017. https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts.

[12] Conway, J. H. *On numbers and games*. AK Peters/CRC Press, 2000.

[13] Corser, G., and Bielby, J. Internet of things (iot) security best practices.

[14] Coskun, V., Ozdenizci, B., and Ok, K. A survey on near field communication (nfc) technology. *Wireless Personal Communications 71*, 3 (Aug 2013), 2259–2294.

[15] Dowling, S., Schukat, M., and Melvin, H. A zigbee honeypot to assess iot cyberattack behaviour. In *Signals and Systems Conference (ISSC), 2017 28th Irish* (2017), IEEE, pp. 1–6.

[16] Editors, D.-K. E. Ensuring device and radio security in low-power iot wireless connections, Aug 17 2017. https://www.digikey.com/en/articles/techzone/2017/aug/ensuring-device-radio-security-lorawan.

[17] Fernandes, E., Jung, J., and Prakash, A. Security analysis of emerging smart home applications. In *2016 IEEE Symposium on Security and Privacy (SP)* (May 2016), vol. 00, pp. 636–654.

[18] Fouladi, B., and Ghanoun, S. Security evaluation of the z-wave wireless protocol. *Black hat USA 24* (2013), 1–2.

[19] Fruhlinger, J. What is wannacry ransomware, how does it infect, and who was responsible?, Sep 27 2017. https://www.csoonline.com/article/3227906/ransomware/what-is-wannacry-ransomware-how-does-it-infect-and-who-was-responsible.html.

[20] Global, I. What is impersonation attacks. https://www.igi-global.com/dictionary/impersonation-attacks/42005.

[21] Group, W. W. P. A. N. W. W. Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2015 (Revision of IEEE Std 802.15.4-2011)* (April 2016), 1–709.

[22] Guarnizo, J. D., Tambe, A., Bhunia, S. S., Ochoa, M., Tippenhauer, N. O., Shabtai, A., and Elovici, Y. Siphon: Towards scalable high-interaction physical honeypots. In *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security* (2017), ACM, pp. 57–68.

[23] HiveMQ. Introduction to mqtt 5. https://www.hivemq.com/blog/mqtt-5-introduction-to-mqtt-5/.

[24] HiveMQ. Mqtt essentials. https://www.hivemq.com/mqtt-essentials/.

[25] HiveMQ. Mqtt essentials: Part 1 – introducing mqtt. https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt.

[26] Hummen, R., Hiller, J., Wirtz, H., Henze, M., Shafagh, H., and Wehrle, K. 6lowpan fragmentation attacks and mitigation mechanisms. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks* (New York, NY, USA, 2013), WiSec '13, ACM, pp. 55–66.

[27] Incapsula. Backdoor attacks. https://www.incapsula.com/web-application-security/backdoor-shell-attack.html.

[28] INCIBE. Security in zigbee communications, Apr 26 2016. https://www.certsi.es/en/blog/security-zigbee-communications.

[29] Jing, Q., Vasilakos, A. V., Wan, J., Lu, J., and Qiu, D. Security of the internet of things: perspectives and challenges. *Wireless Networks 20*, 8 (Nov 2014), 2481–2501.

[30] Kettle, N. Undocumented backdoor account in dbltek goip, Mar 2 2017. https://www.trustwave.com/Resources/SpiderLabs-Blog/Undocumented-Backdoor-Account-in-DBLTek-GoIP/.

[31] Lab, S. Z-wave specification. https://www.silabs.com/products/wireless/mesh-networking/z-wave/specification.

[32] Levis, P., Clausen, T., Hui, J., Gnawali, O., and Ko, J. The trickle algorithm. RFC 6206, RFC Editor, March 2011. http://www.rfc-editor.org/rfc/rfc6206.txt.

[33] Leyden, J. We found a hidden backdoor in chinese internet of things devices, Mar 2 2017. https://www.theregister.co.uk/2017/03/02/chinese_iot_kit_backdoor_claims/.

[34] LoRa Alliance. *LoRaWAN Specification*, 2015. Rev. 1.0.

[35] MalwareMustDie. Mmd-0055-2016 - linux/pnscan ; elf worm that still circles around, Aug. 24 2016. http://blog.malwaremustdie.org/2016/08/mmd-0054-2016-pnscan-elf-worm-that.html.

[36] Manuel, J. Searching for the reuse of mirai code: Hide 'n seek bot, Apr. 16 2018. https://www.fortinet.com/blog/threat-research/searching-for-the-reuse-of-mirai-code--hide--n-seek-bot.html.

[37] Mattern, F., and Floerkemeier, C. *From the Internet of Computers to the Internet of Things*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 242–259.

[38] Mayzaud, A., Badonnel, R., and Chrisment, I. A Taxonomy of Attacks in RPL-based Internet of Things. *International Journal of Network Security 18*, 3 (May 2016), 459 – 473,.

[39] Miorandi, D., Sicari, S., Pellegrini, F. D., and Chlamtac, I. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks 10*, 7 (2012), 1497 – 1516.

[40] Moriuchi, P., and Chohan, S. Mirai-variant iot botnet used to target financial sector in january 2018, Apr. 5 2018. https://www.recordedfuture.com/mirai-botnet-iot/.

[41] OASIS Standard. *MQTT Specification*, 2014. Version 3.1.1.

[42] Organization, Z. S. Zigbee specification. http://www.zigbee.org/wp-content/uploads/2014/11/docs-05-3474-20-0csg-zigbee-specification.pdf.

[43] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., and Rossow, C. Iotpot: Analysing the rise of iot compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT 15)* (Washington, D.C., 2015), USENIX Association.

[44] Patel, K. K., Patel, S. M., and Professor, P. S. A. Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *Int. J. Eng. Sci. Comput 6*, 5 (2016).

[45] Proano, A., and Lazos, L. Packet-hiding methods for preventing selective jamming attacks. *IEEE Transactions on Dependable and Secure Computing 9*, 1 (Jan 2012), 101–114.

[46] Provos, N. Honeyd-a virtual honeypot daemon. In *10th DFN-CERT Workshop, Hamburg, Germany* (2003), vol. 2, p. 4.

[47] Raza, S., Duquennoy, S., and Selander, G. Compression of ipsec ah and esp headers for constrained environments.

[48] Reasearch, C. P. Check point threat alert: Shodan. https://blog.checkpoint.com/2016/01/04/check-point-threat-alert-shodan/.

[49] Reasearch, C. P. Iotroop botnet: The full investigation, Oct. 29 2017. https://research.checkpoint.com/iotroop-botnet-full-investigation/.

[50] Rouch, L., François, J., Beck, F., and Lahmadi, A. A universal controller to take over a z-wave network. In *Black Hat Europe 2017* (2017), pp. 1–9.

[51] Ryan, M., et al. Bluetooth: With low energy comes low security. *WOOT 13* (2013), 4–4.

[52] Schwartz, M. J. Hackers exploit weak remote desktop protocol credentials, Nov 3 2017. https://www.bankinfosecurity.com/hackers-exploit-weak-remote-desktop-protocol-credentials-a-10433.

[53] Seals, T. Bricker bot follows mirai tactics to permanently dos iot devices, Apr. 7 2017. https://www.infosecurity-magazine.com/news/bricker-bot-follows-mirai-tactics/.

[54] Semtech. What is lora®? https://www.semtech.com/technology/lora/what-is-lora.

[55] Shah, A., Pal, A., and Acharya, H. B. The internet of things: Perspectives on security from RFID and WSN. *CoRR abs/1604.00389* (2016).

[56] Shelby, Z., and Bormann, C. *6LoWPAN: The Wireless Embedded Internet*. Wiley Series on Communications Networking & Distributed Systems. Wiley, 2011.

[57] Shirey, R. Internet security glossary, version 2. RFC 4949, RFC Editor, August 2007. http://www.rfc-editor.org/rfc/rfc4949.txt.

[58] Shoemaker, A. How to identify a mirai-style ddos attack, Apr. 10 2017. https://www.incapsula.com/blog/how-to-identify-a-mirai-style-ddos-attack.html.

[59] SIG, B. Core specifications. https://www.bluetooth.com/specifications/bluetooth-core-specification.

[60] TapTrack. Nfc relay attacks, Dec 5 2016. https://www.taptrack.com/article/blog/nfc-relay-attacks/.

[61] Tierney, A. Z-shave. exploiting z-wave downgrade attacks, 23 May 2018. https://www.pentestpartners.com/security-blog/z-shave-exploiting-z-wave-downgrade-attacks/.

[62] Vidgren, N., Haataja, K., Patino-Andres, J. L., Ramirez-Sanchis, J. J., and Toivanen, P. Security threats in zigbee-enabled systems: vulnerability evaluation, practical experiments, countermeasures, and lessons learned. In *System sciences (HICSS), 2013 46th Hawaii international conference on* (2013), IEEE, pp. 5132–5138.

[63] Weinstein, R. Rfid: a technical overview and its application to the enterprise. *IT Professional 7*, 3 (May 2005), 27–33.

[64] Williams, C. Today the web was broken by countless hacked devices – your 60-second summary, Oct. 21 2016. https://www.theregister.co.uk/2016/10/21/dyn_dns_ddos_explained/.

[65] Winter, T., Thubert, P., Brandt, A., Hui, J., Kelsey, R., Levis, P., Pister, K., Struik, R., Vasseur, J., and Alexander, R. Rpl: Ipv6 routing protocol for low-power and lossy networks. RFC 6550, RFC Editor, March 2012. http://www.rfc-editor.org/rfc/rfc6550.txt.

[66] Wright, J. Killerbee: practical zigbee exploitation framework. In *11th ToorCon conference, San Diego* (2009).

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**