Understanding security flaws of IoT protocols through honeypot technologies

Meng Wang

Technische Universiteit Delft

TU Delft

Understanding security flaws of IoT protocols through honeypot technologies

by

Meng Wang

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 29, 2017 at 13:30.

Student number:	4502663	
Project duration:	November 27th, 2016 – August 29th, 2017	
Thesis committee:	Dr. ir. Fernando A. Kuipers	TU Delft, supervisor
	Marco Zuniga, PhD	TU Delft
	Ir. Javier Santillan,	Brightsight B.V. , supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

brightsight®
the number one
security lab
in the world

Preface

This report is the result of my Master Thesis Research, with which I hope to complete my Degree in Electrical Engineering. In this research, I have investigated the security of Internet of Things (IoT) Platform where I proposed a honeypot used for catching attacker's activities against the IoT platform. This research is supervised by both the security evaluation lab Brightsight B.V. and Delft University of Technology. For me, the field of security and honeypot turned out to be a perfect fit with my interests on cyber security. And the topic of Internet of Things is something I wanted to study on. I achieved the goals of getting more knowledge on Internet of Things and Cyber Security through this project.

I would like to express my thanks for the valuable suggestions and continuous support from Dr. ir. Fernando A Kuipers. You are a patient and responsible supervisor.

Many thanks to Mr. Javier Santillan as my supervisor in *Brightsight B.V.* for all the technical support, encouragement and patience. Thanks for giving me inspiration and motivation on this topic to work out such an interesting and valuable project.

I want to extend my gratefulness to my friendly and helpful colleagues from *Brightsight B.V.*. You gave me suggestions and support during this time. I enjoyed and cherished the time we spent together.

My last, and most heartfelt words are to my parents. You are always supporting me and giving me a lot of love.

Meng Wang
Delft, August 2017

Abstract

Internet of Things (IoT) devices are gaining popularity in daily life as well as in specific fields such as home automation, medical facilities, among others. Many applications can be developed in each domain and new ones appear everyday, requiring a flexible, simple and secure interconnection among "things" [38]. Moreover, IoT platforms could integrate devices that have different interfaces and services. When IoT devices such as SmartTV, consoles, media devices, refrigerator, medical devices, etc. are reachable from the Internet, they may be more vulnerable since the security mechanisms of IoT protocols are not yet developed as common systems (e.g. PC, smartphones). The need of the improvement of security mechanisms for IoT devices and platforms is more evident since more related attacks have been seen on Internet ([46]). To achieve the improvement, the Honeypot technology can be used to understand the attackers' behaviour and techniques against emerging IoT technologies. Thus, by analysing gathered data, it is possible to provide feedback to the security domain of IoT devices by detecting and analyzing attack vectors. Results can be used to interpret the impact of such trends within the context of not only IoT devices themselves, but also to the whole IoT platform.

In this thesis, a literature study of current technologies for IoT platforms is performed, focusing on IoT security mechanism. This research includes analysis of IoT application and communication protocols such as MQTT, XMPP, HTTP REST, AMQP, CoAP, UPnP, JMS. Moreover, a novel IoT honeypot, ThingPot, is proposed to study the security problems of an IoT platform. As far of the findings of the literature review, this honeypot is the first of its type since it is focused not only the application protocols themselves (such as IoTPOT [53], Telnet IoT honeypot [55], etc.), but on the whole IoT platform. A Proof of Concept (PoC) is implemented with XMPP and HTTP REST through the use case Philips Hue smart light IoT system. By analyzing the collected data, we find five main kind of attacks against smart devices and conclude the pros and cons of XMPP on IoT platform in terms of security. Findings also provide feedback about how a honeypot for IoT platforms can be deployed.

Contents

1 Introduction

Today, the concept of Internet of Things (IoT) is very popular. A growing number of IoT devices appear at an unprecedented rate. Big companies such as Amazon, IBM, Samsung, etc. have built IoT related projects and some products (platforms, mobile applications, etc.) have been launched. Even Ikea recently launched their Trådfri smart lighting platform in the US [30]. Also many new companies are focusing on IoT technologies. "IoT" has been a familiar term for us and are gaining increasing attention from both the market and the citizens. There is a prediction that by 2020, tens of billions of things will be deployed worldwide, collecting a wealth of diverse data [31].

What is the Internet of Things? As the name indicates, the Internet of Things means that "things" are connected to each other via the Internet. The concept of "Internet of Things" might be first proposed by Kevin Ashton on 1999 according to the speech [14]. Then the ITU released a report about IoT "Internet of Things" [66] in 2005, introducing IoT in terms of enabling technologies, market, opportunities and challenges. Till now, IoT has become a globally famous concept. It is not only a technical word, but also it has been applied to economy. It enables physical objects to see, hear, think and perform jobs by having them "talk" to each other, to share information and to coordinate decisions.[12] For example, by combining the smart sensors in heating, light, air conditioning and humidity at home to build a monitoring and control system through the Internet. The curtain can be opened and closed automatically. Coffee will be prepared when you wake up. You have a smart TV, smart temperature control system, etc. This "smart home" can be seen as an IoT application. The IoT technology can be applied in various fields such as medical facilities, Industrial automation, transportation etc. It is expected to increase quality of life and growing the world's economy.

The Internet of Things involves many fields and many kinds of technologies such as embedded devices, communication technologies, sensor networks, Internet protocols, etc. Survey papers that address different aspects of IoT technologies have been published. For example, [12] provides an overview of the Internet of Things with emphasis on enabling technologies, protocols, and application issues. In [63], an overview of current standards and research activities in both industry and academia is presented. The author of [31] provides main technological components needed to realize the IoT concepts and applications. [15] covers both the main communication enabling technologies, and the wired, wireless and the elements of wireless sensor networks (WSN).

An IoT system should be able to handle heterogeneous objects through the Internet. This represents a critical requirement of the IoT architecture. In general, the IoT architecture can also be layered, however, the definitions and classification of each layer are not standard and vary in different literature. The survey [12] summarized all the proposed models, making a conclusion that in the pool of all the proposed models, the basic model is a 3-layer architecture consisting of the Application, Network and Perception Layers. However, a 5-layer model is also proposed and has been used. The five layers are: Objects Layer, Object Abstraction Layer, Service Management Layer, Application Layer and Business Layer.

No matter how many layers there would be, the IoT architectures are in some terms similar with traditional Internet related devices. This means, they are communicating with each other through protocols on each layer. Protocols are the language of network devices. In the IoT context, "things" have different capabilities and characteristics. The heterogeneousness requires the protocol to fit more kinds of devices, architectures and situations. Moreover, the smart sensors are resource constrained and often can not do complex calculations. Thus there are new requirements for IoT communications. Some existing protocols that fulfill the IoT requirements are adapted and utilized on different layers in IoT. To take the application layer protocols as an example, XMPP, CoAP, HTTP, etc. are widely used in IoT products. However, these protocols were designed for other purposes, such as real-time communication, asynchronous communication, messaging, etc.

With the increasing of Internet of Things, research on IoT has increased as well, focusing on the IoT protocols. [31] provides a picture of the main technological components needed to enable the interconnection among "things" in order to realize IoT concepts and applications. [12] provide a thorough summary of the most relevant protocols and application issues. It also presents detailed service use-cases to illustrate how the different protocols presented in the paper fit together to deliver desired IoT services. [37] presents and compares existing IoT application layer protocols as well as protocols that are utilized to connect the "things", and also end-user applications to the Internet. Moreover, websites that provides introduction and implementation of these protocols were built [35] [71].

However, IoT is still in its initial stage and therefore it is not mature enough. A consensus has not been reached completely on the entire IoT ecosystem, both in industry and academy.

1.1. Motivation and Related Work

1.1.1. Demands of the market

Internet of Things (IoT) devices are gaining popularity in daily life as well as in specific fields such as medical facilities. Their usage can be done through a local network or Internet. When IoT devices such as SmartTV, consoles, media devices, refrigerator, medical devices, etc. are reachable from Internet, they may be more vulnerable since the security mechanisms of IoT protocols and applications are not yet developed as common systems (e.g. PC, smartphones).

Taking the Industrial IoT (IIoT) as an example, according to [40], a survey involves 403 IT professionals who hold some responsibility for digital security as a significant part of their job. It identified the extent to which their organizations are prepared to face IIoT-related threats in 2017. The results of the survey are: Firstly, when asked about the coming year, 96 percent of respondents said they expect to see an increase in security attacks on IIoT; Secondly, 51 percent said they are not prepared for malicious campaigns that in some way exploit or misuse the Industrial Internet of Things; Finally, 64 percent of participants already recognized a need for their organizations to protect against such attacks.

1.1.2. Vulnerable and immature platform

As mentioned before, since things are connected to the Internet, the traditional security problems (e.g. virus infections, hackers, etc.) may also impact the IoT communication. Moreover, because of the features including heterogeneity, energy conservation and automation, the Internet of Things is facing more security problems. If the devices or the software have problems, there might be serious consequences. According to [43], 1.6 million vulnerable Internet-connected things have been found, which are passwordless login or login using default credentials. About 420,000 of these vulnerable devices were comprised by the "Carna Botnet".

Some of the largest distributed denial-of-service (DDos) attacks ever recorded appeared in 2016, supposedly reaching up to approximately 1 Tbps of traffic and performed by hundred of thousands of compromised IoT devices. In this case, infection of IoT devices is known to be by the so-called malware Mirai [46], which puts vulnerable IoT devices within the control of a Botnet and which is believed to control about a million of compromised IoT devices. Security incidents like this might be just the beginning within the context of IoT.

1.1.3. Lack of research

Currently, there are no standards that define in a consistent way the usage of protocols within IoT platforms. At different layers, multiple protocols can be used, some of which are either under development or are actually designed for other or more generic purposes. Moreover, existing surveys show a convergence towards the usage of XMPP, MQTT, CoAP, REST APIs, etc. for wide deployment of IoT platforms.

A number of surveys about IoT security in different perspectives are done by researchers. [32] analyzes existing protocols and mechanisms to secure communications in the IoT, as well as open research issues. They also analyze how existing approaches ensure fundamental security requirements and protect communications on the IoT, together with the open challenges and strategies for future research work in the area. The analysis focuses on the IoT communication protocols , divided into four main parts: physical (PHY) and Medium Access Control (MAC) layer, network layer, routing in the IoT (RPL), and application layer. In [29], the application security for the different security profiles is also elaborated. [42] and [28] provided an overview of security issues and gave some advices on improving security.

To summarize, security research findings have identified issues that include:

- Perception layer security issues:
  - Key bootstrapping approach problems in IoT (Key distributions, Key management, etc. [47][70]
  - Public key cryptography vs private key operations [70]
- Network Layer security issues:
  - Network's DoS attacks to sensor network nodes [28]
  - Sensor network gateway nodes out of control [28]
  - Interference, tampering and destruction to the network signal
  - Security Issues in End-to-End Handshake [17]
- Application layer security issues:
  - Limitations of CoAP Security[32]

[56] lists six trending technologies for IoT security based on Forrester's analysis. These are:

- network security
- IoT authentication
- IoT encryption
- IoT PKI
- IoT security analytics
- IoT API security

It can be noted that most of the surveys are mainly focusing on encryption and authentication issues on the network layer. The application layer's security problem is also serious and we need to pay special attention. Unfortunately, although there are a lot of surveys about IoT protocols, there is a lack of research on the application protocols' security within the IoT context, which implies a gap that may include vulnerabilities that cannot be ignored. For example, the attack tool XMPPloit [22] shows that there are existing vulnerabilities of XMPP which can also be exploited in an IoT context. Thus it is necessary to research IoT application protocols vulnerabilities and attacks.

1.2. Vulnerability analysis on IoT

A weakness may be not only on the protocols themselves, but also on their implementation. There might be problems when implementing a server, a client. Common Vulnerabilities and Exposures (CVE) records the known weaknesses. Attackers can also make use of the weaknesses that have not been fixed or to attack implementations that are not patched on time. So it is necessary to learn about vulnerable implementations and related attack vectors. There are different approaches to perform a vulnerability analysis on IoT:

- Protocol analysis:
  Some researchers like [64] chose to analyze the protocol itself. Because of the heterogeneous nature of IoT, the usage of protocols is complex and varies. There are challenges to security testing and analysis of IoT protocols. Moreover, IoT technology is not only related to protocols, the deployment and integration between protocols and technologies should also be taken into consideration while doing the vulnerability analysis. Tools like Proverif can also be used.

- Fuzzing testing:
  By providing invalid, unexpected, or random data as inputs the attacker might get useful data. Fuzzing can test if the protocol or an implementation is secure enough.

- Honeypot:
  Honeypot is a computer security mechanism that can track attacker's activities, through which can improve the security level and might solve some of the security problems of IoT. Basically its purpose is to be attacked within a monitored and controlled environment. By implementing a honeypot with IoT protocols, the attackers will try to attack it, thus the honeypot will record the attacker's information (e.g. IP address, attacking tools). Such information will help to study the security problems of IoT.

Comparing these three ways of vulnerability analysis, in this thesis, we take the approach of using honeypot technology as the method to analyze and study the IoT communication security problems. Based on the the current situation, an IoT Honeypot, ThingPot is going to be designed and implemented to study on the IoT security problems on application layer. It will be an simulation of IoT platform instead of single communication protocols, which makes it fit the current IoT products situation and explore deeper about the security problems.

1.3. Contribution

The main goal of this research is to answer the question ***"What are the existing weaknesses of common IoT protocols"***. The approach to answer this question is through the use of Honeypot technologies, which, by performing protocol emulation and data analysis, may be able to identify weaknesses from actual compromises (i.e. exploitation) of vulnerable IoT devices within a controlled environment. With this regard, the following support questions are defined:

- What are the related protocols used by IoT devices?
- What are the existing vulnerabilities and the nature of known attacks on IoT protocols?
- How can Honeypot technologies be implemented in order to be accurate, efficient and be able to identify security trends on IoT devices?
- Are honeypot technologies suitable to identify security issues on IoT platforms?
- Based on the findings on IoT protocol security, what can be done to prevent attacks on IoT devices?

1.4. Methodology

In order to achieve the goal of the research, the following steps shall be followed.

- Conduct a research on common IoT related protocols such as MQTT, XMPP, XDS, AMQP, UPnP to get and understand their design. This survey will provide a framework to focus on specific protocols and potential research subtopics.
- Identify potential weaknesses of the aforementioned protocols. Perform a literature review to gain knowledge about known security issues.
- Understand Honeypot technologies, their goal, design features and potential advantages for IoT protocol implementation. Existing honeypot technologies for both, common platform and IoT, will be reviewed in order to identify potential improvements ways and suitable paths for new IoT honeypot implementations.
- Once the theoretical framework is defined, a proof-of-concept (PoC) of an IoT honeypot shall be implemented. Such PoC is comprised by:

  - A selected protocol implementation through emulated services (i.e. Honeypot practical implementation [Python/Perl based])
  - The network/server setup for the honeypot deployment. These honeypots will be implemented on Raspberry Pis (a credit card sized ARM Linux box).

- Develop and deploy data analysis mechanism to collect and process logs generated by the honeypot (i.e. network traffic, activity logs, protocol behaviour).
- Analyze the results and get conclusions based on the defined research questions.

1.5. Thesis Outline

In this thesis, there are 6 chapters. In the first chapter, introduction of Internet of Things and the motivation of the thesis are elaborated. Then Chapter 2 introduces the basic knowledge of IoT application protocols and the situations about security of IoT and the IoT platforms. In chapter 3, honeypot technology and the reason of using honeypot is introduced. ThingPot, an IoT platform honeypot is proposed in this chapter too. A Proof of Concept implementation with use cases are described in Chapter 4 to show how ThingPot works. After the implementation, Chapter 5 performs data analysis based on the dataset got in the ThingPot Implementation. Results and findings of the analysis are also shown in this chapter. Then in the last chapter 6, conclusions an future works are given.

2

IoT application protocols

In this chapter, IoT application protocols and related security considerations will be discussed.

2.1. Application protocols of IoT

Communication protocols define a standard way for two or more entities to establish a meaningful interaction which allow a valid, legitimate and expected behavior of all the involved parties. In section 2.1.1, the protocols that are mainly used in the IoT application layer will be generally introduced. Subsequently the details of XMPP, MQTT will be elaborated.

2.1.1. Introduction of IoT application protocols

Since there are no clear and well defined standards for all of the Internet of Things components, the technologies currently used by IoT platforms have different characteristics. Basically, any technology that fulfills the connectivity requirements can be used. In the current situation, different companies have their own IoT architecture. To summarize, there are three main ways of setting up an IoT network for devices to reach each other and the Internet:

1. The devices can connect to each other by using various indoor radio technologies, such as Wi-Fi, Zigbee or Bluetooth. A hub or gateway will be needed to connect to the Internet. It will collect the information from the devices and then transfer to the end users through other application layer protocols. This pattern is often used in smart home or medical facilities.

2. The devices can also use cellular technologies such as 2G/3G/LTE or 5G. Then they can be connected through a gateway. These technologies are fit for outdoor devices own a long distance. But these technologies consume more energy. Thus new long distance technologies are promoted for the IoT. Long Range Wide Area Network(LoRaWAN) and Narrow Band IoT (NB-IoT) are two dominant technologies that are released by different vendors. This method is often used in large scale scenarios such as smart city.

3. Instead of deploying a gateway, the devices can also be directly connected to the Internet. For example, by using the MQTT protocol, the sensors can publish information to the MQTT broker. Then the users can reach the information by connecting to the broker.

Regardless of the kind of wireless technology used, the end-devices' data may be available on the Internet. This can be achieved in two ways: 1) sending information to a proprietary web service or Application Programming Interfaces (API) that is accessible from the Internet; 2) employing the cloud [37]. These web service or API or cloud will be the data base to store and process, the intermediate node between the devices and end user and the API to make the end user able to monitor and control the devices remotely. For example, the Philips Hue product is using Zigbee to connect the end devices (lamp, sensor) and the "Bridge"(Hub). Then the Bridge connects to the Philips server. The end user could control and receive data through the server by using a REST API.

In this thesis, the application protocols that are used to handle the communication between the gateways, and from the gateways to the end users will be studied and discussed.

Many application protocols have been found suitable for IoT communication. Here is a list of related application protocols:

- **MQTT: Message Queue Telemetry Transport**
  MQTT is a messaging protocol and was released by IBM in 1999. It is light weight, open, simple, and designed so as to be easy to implement. It uses the publish/subscribe pattern that runs on top of TCP, so the client does not require updates. Also it has low overhead compared to other TCP based application layer protocols[69]. These features make it light weight and meet the requirement of IoT. It also provides three-level Quality of Service(QoS).

- **XMPP: Extensible Messaging and Presence Protocol**
  XMPP is a communication protocol that provides basic instant messaging (IM) and presence functionality. It was standardized by the IETF and has been widely used. It gained more attention recently due to its features that are suitable for IoT. XMPP is extensible since it allows the specification of XMPP Extension Protocols (XEP) to increase functionality. New XEPs have been released to support IoT. It also uses the publish/subscribe pattern. Nowadays a lot of products support XMPP for customization. More details will be introduced in the next section.

- **AMQP: Advanced Message Queuing Protocol**
  AMQP is an open standard application layer protocol that arose from the financial industry. It uses publish/subscribe communication model and supports reliable communication by utilizing primitive types including at-most-once, at-least-once and exactly once delivery. It has been reported that an AMQP environment with 2,000 users spread across five continents can process 300 million messages per day[26].

- **CoAP: Constrained Application Protocol**
  CoAP is an application layer protocol that was created by Constrained RESTful Environment (CoRE) working group[62][16]. It was designed targeting constrained-recourse devices. It is a request/response protocol that runs over UDP to keep lightweight and reduce bandwidth requirements. But it supports QoS and uses a simple Stop-and-Wait retransmission mechanism for confirmable messages[37]. CoAP uses a simple and small format to encode messages[12].

- **UPnP: Universal Plug and Play**
  UPnP is a set of network protocols that are used for discovery of network devices for data sharing, communications and entertainments. It is a distributed, open architecture protocol based on established standards such as the Internet Protocol Suite (TCP/IP), HTTP, XML, and SOAP. It is used in IoT because it can dynamically enable robust connectivity between devices from different vendors. Thus, it is suitable for home automation.

- **JMS: Java Message Service**
  JMS is an Application Programming Interface that is used to build asynchronous communication between applications or distributed systems. It has been widely used and supported by enterprises. More than two clients can exchange messages reliably and asynchronously.[31]

- **HTTP REST: HTTP REpresentational State Transfer**
  REST is an architectural style instead of a single protocol that was developed by Roy Thomas Fielding in 2000[27]. It has been widely used in Machine-to-Machine(M2M) and IoT platforms. It provides a resource-oriented messaging system, where the resources are accessible via URI and a GET request, whereas inputs are accepted via PUT[31][37].

- **DDS: Data Distribution Service**
  DDS is a middleware protocol from the Object Management Group(OMG)[50] that lies between the operating system and applications. It uses publish-subscribe pattern for data exchanges that is scalable, low-latency, dependable, high-performance and interoperable. It could be used for IoT applications.

Each of these protocols have unique characteristics and can be used in different scenarios. They could also cooperate with each other by being implemented in different parts in an IoT system. Comparisons have been done in many surveys. Brief introductions of the protocols can be found in Karagiannis et al. It also argues their suitability for the IoT in terms of reliability, security and energy consumption. [12] gives a more thorough summary of most the relevant protocols. It enables readers to get to know how different protocols can work together without the need to go through RFCs. Table 2.1 is a summary of the properties of the main IoT communication protocols.

Taking smart home as an example to elaborate the different usages of different protocols: the control of smart light can use XMPP; for the electric power, DDS could be used to monitor the engines in the power plant; MQTT could be used for inspection of the power cable; all the data of the power consumption of the electric devices at home could be transmitted by AMQP to the cloud or home gateway to analyze;

	MQTT	XMPP	AMQP	CoAP	UPnP	JMS	HTTP REST
Transport	TCP	TCP	TCP	UDP	UDP	Not specified	HTTP
QoS Options	Yes	No	Yes	No	Yes	Addressed by W3C WG	No
Architecture	Pub/Sub	Pub/Sub or Req/Res	Pub/Sub	Req/Res		Req/Res	Req/Res
Real-time	No	Near Real-time	No	Near Real-time	No	No	No

Table 2.1: Properties of IoT communication protocols

2.1.2. XMPP

XMPP is an instant messaging protocol originally developed by the Jabber open source community. It used to be called Jabber and it works over TCP. It makes use of a so-called Jabber Identifier (JID) which is a unique identifier for each device. This JID comprises by three parts: node, domain and resource and its format is similar to the email address, which is *"node@domain/resource"*. The node and domain is separated by an "@" and after domain with a slash if there is a resource name. A JID without a resource is called "bare JID".

The XMPP is using a model of "client-server-server-client" as Figure 2.1 shows. The servers are connecting to each other. The clients under different servers are not connected directly. When clients under different servers want to connect to each other, all the communications are through the server in between. By default, the client connects to the XMPP server using the port TCP/5222. And the XMPP servers connect to each others through the port TCP/5269.

The connection between the XMPP client and server relies on a stream of XML(eXtensible Markup Language) stanzas. Communication starts by an opening XML <stream> tag and ends by a closing XML </stream> tag. During the life of a stream, child elements could be sent to exchange information. Stanza is the structured element that can be sent during the life of a stream. There is no limitation of the amount of sending stanzas. An XMPP stanza has different attributes: five common attributes are "to", "from", "type, "id", "xml:lang".[52]. It consists of three components[12][52]:

- Presence:
  The presence stanza is used to notify and update the status of an entity to the entities that have subscribed to it.

- Message:
  The message stanza contains the subject and body fields which are filled by the title and content of the message. It is seen as a push mechanism that one entity pushes information to other entities. The server who receives the message then sends it to the intend recipient according to the "to" attribute in the message.

- Iq(info/query):
  "Iq" is seen as a "request-response" mechanism. One entity can send as "iq" with the type of "get" or "set" to make a request. The recipient can reply with an "iq result" or " iq error".

XMPP is widely used in industry. There are a lot of XMPP servers such as ejabberd, Tigase, Openfire, jabberd, etc., for various kinds of operating systems.

Because of the features of XMPP, it caught the attention of IoT researchers. XEP-0323(IoT-Sensor Data), XEP-0324(IoT-Provisioning and Security), XEP-0325(IoT-Control) and XEP-0326(IoT-Concentrators) are the XEPs designed for IoT. Academic papers[37][12][31] argue that XMPP is a suitable protocol for IoT. Also, there are a lot of use cases for XMPP in IoT. For example, IoTfy is a company that provides end-to-end IoT solutions for industry. It supports XMPP to transfer messages using their own XMPP servers through the domain "xmpp.iotfy.co".

XMPP performs excellent in home automation. A lot of developers prefer to use XMPP as the protocol for their home automation project[73][2]. It almost provides all the functions that a home automation system requires for a messaging protocol, e.g. peer to peer connection, routing to a user, user authentication, publish, subscribe, etc. Existing products like Philips Hue can also be combined perfectly with XMPP in such a way that it is compatible with other products facilitating home automation systems.

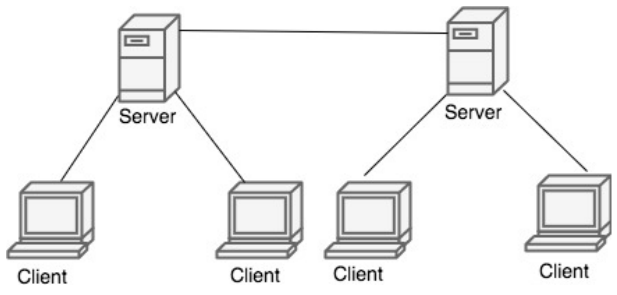


Figure 2.1: XMPP Communication

2.1.3. MQTT

MQTT is a publish/subscribe message transporting protocol that is light weight, open and simple to implement. It is considered to be suitable for M2M and IoT.

The elements in an MQTT communication can be divided into client and broker, which is a server. The clients can be divided into two kinds according to the roles in a communication: the publisher which sends particular messages with a topic and the subscriber which receives particular messages with a topic. One client can be both publisher and subscriber in different topics. The broker is the one in the middle that receives every message and connects to the clients. Its job is to filter the messages and distribute them accordingly based on the topic. Figure 2.2 shows an example of how the MQTT broker and clients work together to transport messages. In this way the sender and receiver do not need to be "online" at the same time.

MQTT also supports three levels of QoS: at most once, at lease once and exactly once.

2.1.4. REST

REST has been widely used in Machine-to-Machine(M2M) and IoT platforms for transferring messages and control commands between the user interface and devices system. Thus the data in the device system could be reached by other Internet applications. A lot of IoT products are using REST API to transfer information between devices and human user. Some examples are:

- Philips Hue uses a REST API for the users to send commands to the Hue bridge. Then the bridge controls the lights according to the commands or sends information back.
- Xively also supports HTTP to publish messages.
- Belkin's Wemo switch supports REST API for the users to send a request to the switch via a url.

The type of HTTP requests includes:

- POST
- GET
- HEAD

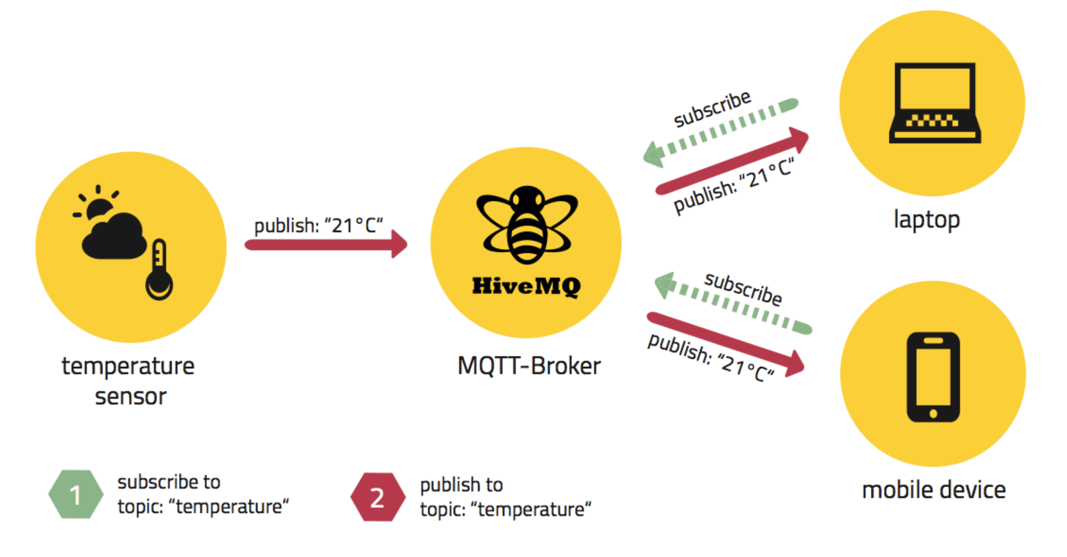


Figure 2.2: MQTT Communication[36]

- PUT
- DELETE
- OPTIONS
- CONNECT

2.2. Security of IoT application protocols

2.2.1. Current situation

The security of IoT is gaining more attention from researchers as well as industries, nevertheless, the security of IoT is still not mature. We can consider the security problems from three aspects: flaws of protocols themselves, problems during protocol implementation, vulnerabilities during the integration with IoT. While implementing the protocols with the IoT platforms, the security mechanism of the protocols themselves is used. Taking a look at the communication protocols that were mentioned before, table 2.2 summarizes the security mechanisms that each communication protocols apply.

MQTT	Simple User-name/password Authentication, TLS/SSL for data encryption
XMPP	SASL authentication, TLS/SSL for data encryption
AMQP	SASL authentication, TLS/SSL for data encryption
CoAP	DTLS/IPSEC
JMS	Vendor specific but typically based on TLS/SSL. Commonly used with JAAS API
SOAP	Address by WS-Security

Table 2.2: Security mechanisms of IoT communication protocols

2.2.2. Current problems of IoT communication security

As has been mentioned in the last section, the security problems can be considered from three aspects:

1. The vulnerabilities of the communication protocols themselves.

   For example, the attack tool XMPPloit[22] shows that there are existing vulnerabilities of XMPP. Here is a list of potential security problems of these protocols.

   - **For all protocols:** Secure booting, firewalling and secure updating and patching problems for all protocols.

[59] lists the security challenges that the IoT might face, the process of secure booting, firewalling and secure updating and patching might have some problems. [47] gives an example: only authorized software should be allowed to be downloaded to the embedded device. Thus relative measures like digital signature or keys for encryption are needed. While considering the large deployment and the limited resources, it will be a security challenge.

- **CoAP:** Problems of using DTLS for CoAP.
  [37] says that DTLS is not designed for CoAP, thus there might be problems. For example, DTLS does not support multicast. [32] listed the limitations of applying DTLS in CoAP and some proposals to solve these problems. But these proposals might not be that appropriate for CoAP and cause security problems. Further research is needed.
- **CoAP:** Problems of CoAP's "NoSec Mode".
  [32] introduces 4 security modes, where the first one is "NoSec Mode". In this mode the CoAP message is transmitted without any security mechanism implemented.
- **MQTT:** Problems of MQTT encryption and authentication.
  [61] says that there is a new protocol called Secure MQTT[64], which uses encryption based on "lightweight attribute based encryption". Differences between sMQTT and MQTT might be the weakness of MQTT.
- **XMPP:** Problems of SASL authentication.
  There might be potential problems of SASL authentication. [44] explains the security consideration that many of existing SASL mechanisms do not provide adequate protections. Extra protective services might be needed to protect against attacks. And XMPP is using SASL for authentication. So any weakness of SASL will be the weakness of XMPP.
- **UPnP:** Famous UPnP weakness.
  The weakness of UPnP is not a new topic that there have already been a lot of discussions about it both on the Internet[45][19] and in academic area[48][13]. [45] mentions that security vulnerabilities in UPnP continue to crop up and continue to put millions of home networking devices at risk for compromise. [19] also explains how to make use of the vulnerabilities of UPnP to execute attacks.

2. Protocol implementation issues

   While implementing the protocol (e.g. build a server and install it), there might be security issues related with development (i.e. bugs, weaknesses, lack of validations, etc.) which open vulnerabilities into the whole implementation. Eventually, these vulnerabilities may be mapped into the Common Vulnerabilities and Exposures (CVE) database. CVE contains a list of the known vulnerabilities for software products including Operating Systems, libraries, frameworks, etc. including both open and closed source implementations. There are different databases that share a common list of CVEs identifiers; for example, CVE details[1] and NVD NIST databases[4]. These databases describe all the available information about the vulnerability, such as vendor, product, time, vulnerable versions, vulnerability type, description, etc. Thus, by searching the CVEs of the IoT related protocols, it is possible to identify exploitation vectors for IoT applications or platforms using those protocols. Developers should upgrade and find solutions to prevent the exploitation of any known vulnerabilities. Some examples of related CVEs are:

   - CVE-2014-2746: known as "xmpp bomb" where an attacker could send a valid compressed XML element with a lot of white spaces, and cause a Denial of Service (DoS).
   - CVE-2016-9877: MQTT connection authentication with a username/password pair succeeds if an existing username is provided but the password is omitted from the connection request.
   - CVE-2010-0305: Some version of ejabberd (XMPP server) allows remote attackers to cause a denial of service (daemon crash) via a large number of c2s (aka client2server) messages that trigger a queue overload.
   - ...

3. Vulnerabilities during integration with IoT
   An IoT platform integrates different IoT technologies and protocols to work together. It has been discussed that IoT is still an immature technology as platform, which is prone to multiple security attacks.

These security problems happen during the integration of different IoT application protocols and this might be just the beginning within the context of IoT. Thus, different approaches can be implemented in order to collect, analyze and eventually identify any pattern about threats of IoT platforms.

2.3. Current Situation of IoT Devices and Platforms

Currently the IoT devices and platforms are booming. People would like to buy the smart devices from different companies and build up a customized smart home system. This personal smart home system usually contains a device control panel, some smart devices and devices' Internet access (Hub, bridge, etc.). On such IoT platform, users can integrate devices from different companies together.

One of the common current IoT platform's framework is as shown in Figure 2.3. It includes an API to work and communicate with the devices, instant communication protocols to build communication between users and API, and the clients (users) could reach both via the API and instant communication protocols. Among the many ways of building up a customized IoT Platform, XMPP protocol is one of the most famous communication protocols that are used. There are a lot of examples of this infrastructure like Wemo, Philips Hue, IoTfy, etc. An example of this framework is the Philips Hue system combined with XMPP by users. Details are introduced in section 4.2.

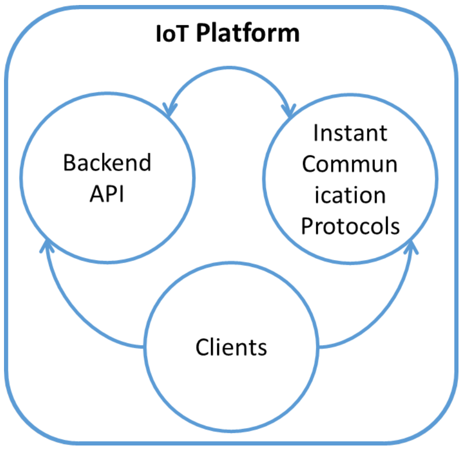


Figure 2.3: Current Situation of the IoT Platform

3 ThingPot: An IoT Honeypot

In this chapter, ThingPot, an IoT Honeypot will be introduced, including an overview about Honeypot technology and the design of the IoT honeypot.

3.1. Introduction

Communication protocols define a standard way for two or more entities to establish a meaningful interaction which allow a valid, legitimate and expected behaviour of all the involved parties. For the case of IoT, protocols in the application layer can define not only the way of how IoT devices can exchange information, but also how they can be managed as part of an IoT platform.

Taking into consideration that protocols define a baseline for expected behaviour, the processes of collection, analysis and identification of anomalies (that may be part of malicious activities) can be performed through dynamic protocol analysis including payload (content) processing, context and general pattern analysis. For this purpose different approaches could be implemented, for example:

- Traffic analysis (Intrusion Detection Systems, Flow analysis, anomaly detection system)
- Honeypot technologies (emulation or deployment of communication protocols running through services (software) that act as decoy systems or traps for intruders, automated malware or any source of malicious or anomalous activity).
- Log analysis of any system involved during the communication (servers, applications, operating systems, etc.)

3.2. Honeypot technology

Honeypot technology is a mechanism that catches the attacker's activities by simulating a real system but placed in a protected environment. The attacker reaches the honeypot when it is seen as a real system or device. Since the honeypot is implemented in a protected and monitored way, the attacker's activities will not really damage the system, while the information of the attacker is recorded. The main concept of honeypot technology is the emulation or deployment of communication protocols running through services (software) that act as decoy systems or traps for intruders, automated malware or any source of malicious or anomalous activities.

3.2.1. Advantages of Honeypot technology

The effectiveness of approaches to be implemented is based on the context, type of network, restrictions or limitations of how IoT devices are deployed within the network. All the three mentioned alternatives could provide useful data to analyze, however this research aims to explore the use of honeypots due to these three main reasons:

Honeypots are a proven way to collect data about:

- Attacks patterns (traffic statistics, trends on attack vectors)
- Malware samples and suspicious binary data
- Potential weaknesses on protocol implementation
- Amount of data is not necessarily an indicator of useful datasets collection

Honeypots take advantage of emulation to address:

- Security measures and more controlled/limited environment
- Lack of actual devices and thus flexibility to cover more contexts

IoT is still in early development stages, thus Honeypots could help to understand their development from the security point of view from early stages such that:

- Findings could give information to develop detection mechanisms (e.g. IDS signatures, ACLs)
- Support the generation of sample references (i.e. traffic/models) that could be used for further honeypots or any detection or countermeasure mechanism.

3.2.2. Approach to honeypot technology and link with IoT platforms

By learning the vulnerabilities mentioned in section 2.2.2, the honeypot can make use of the information and simulate the vulnerable version of the implementation. In this way it might attract the attackers to execute attacks. And thus catch the attacker's activities and information.

This section summarizes an approach about how honeypots could be applied in IoT platforms including general description, advantages and previously known considerations that might represent issues during the research, development and experimentation process.

By using honeypots, not only IoT devices can be emulated, but also a whole IoT platform that provides access control and management systems for IoT devices. Emulation can be done at multiple layers and scopes such that the attacker can gain a wider level of interaction and therefore more information could be analyzed.

- Device emulation can be achieved through deployment of both actual and emulated devices providing fake data and features (e.g. motion sensors, temperature, smart lights, etc)
- Platform emulation can be achieved through deployment of both actual and emulated servers (e.g. XMPP, MQTT, REST API backends serving IoT management)

The level of interaction can be defined as the range of possibilities that a honeypot allows an attacker to have. In general, there are three types of honeypot in terms of the level of interaction. ThingPot is a hybrid honeypot.

High Interaction Honeypots (HIH)
HIH are basically real systems which use standard protocol implementations. Its main feature is that being a real system, then full interaction with attackers can be performed. However, there are security concerns to be addressed since actual exploitation could happen. Thus, HIH involve the deployment of monitoring and network control systems that keep the environment secured during attackers activities.

IoT platform could be implemented through deployment of XMPP/MQTT/REST HIH honeypots. Being actual systems, attacker could interact from the starting point. Further emulation (i.e. interaction with devices) can be implemented through Low Interaction Honeypot.

Low Interaction Honeypot (LIH)
Low Interaction Honeypots detect attackers using software emulation of characteristics of a particular operating system, application, network services or protocols on top of a host operating system. Advantage of this approach is to get better control over attacker's activities and less security risks since attacker is limited to emulated features, which under actual exploiting conditions, attacks will not work and still they will be

recorded. On the other hand, disadvantage about this approach is the fact that the low-interaction honeypot emulates services, or steps within a protocol, but it may not emulate the complete design or features of such applications or protocols which might also limit data acquisition and interaction with the attacker. Examples of this type of honeypots are Dionaea[57], Honeyd[58], NetBait[20], Kippo[51], etc.

IoT device emulation can be done through LIH. Moreover, interaction with both real and other emulated XMPP and existing MQTT services can be used.

Medium Interaction Honeypots (MIH)
Honeypots that offer attackers more ability to interact than the low-interaction honeypots, but less functionality than high-interaction solutions, are called medium-interaction honeypots. They can expect certain activity and are designed to give certain responses beyond what a low-interaction honeypot would give. MIH combine features of both LIH and HIH, however they might involve more complexity to design and deploy.

The prototype of the proposed IoT honeypot actually involves a MIH. However, it is not meant to include the full design feature for the honeypot to work, since early stages and independent module emulation (i.g devices) can be enough to gather data for further analysis.

Table 3.1 lists the pros and cons of HIH, LIH and MIH. It could be seen that the HIH is in hight complexity, which costs more time and efforts, while it catches more information at the same time. The LIH can only detect mature exploits. In this thesis, both the HIH and LIH will be used.

	pros	cons
HIH	1. Provides more attractive environment for interaction 2. Deeper interaction with the attacker 3. Catch more data and gain more information	1. High complexity of implementation and maintain 2. Higher risk to be broken into and controlled by the attacker
LIH	1. Low complexity of implementation and maintain. 2. Lower risk to be broken into and controlled by the attacker. 3. High scalabilty 4. Do not require significant computing mechanism	1. Limitation of the data that it can capture. 2. No deeper interaction with the attacker
MIH	1. Provides balance of cost and gained value of data. 2. Higher ability than LIH	Higher costs than LIH

Table 3.1: Comparison of HIH, LIH and MIH

3.2.3. Existing Work

There are only a few known implementations of honeypots that are focused on the Internet of Things:

- Telnet IoT honeypot [55]: This is a honeypot that implements a telnet server in python to catch the IoT Malware.
- HoneyThing [49]: This is a honeypot that is designed for TR-069 (CPE WAN Management Protocol) protocol, which is for the Internet of TR-069 Things.
- IoTPOT [53]: A novel honeypot to emulate Telnet services of various IoT devices to analyze ongoing attacks in depth. IoTPOT consists of a frontend low-interaction responder cooperating with backend high-interaction virtual environments called IoTBOX. IoTBOX operates various virtual environments commonly used by embedded systems for different CPU architectures.
- Dionaea [24] [57]: A honeypot framework that, among others, implements a MQTT module.
- ZigBee Honeypot [25]: A honeypot that simulates a ZigBee gateway.
- Multi-purpose IoT honeypot [39]: An IoT honeypot that focuses on Telnet, SSH, HTTP and CWMP.

3.3. ThingPot Design

3.3.1. Design Decisions

Taking all the research into consideration, to find out the problems of current IoT platforms, a honeypot that simulating IoT Platforms is a a good tool to solve this question. In this thesis, a honeypot, ThingPot is proposed. It is designed to simulate the IoT Platform that is well-known currently, aiming at the problems that are exposed now or potential problems that is still unknown.

In this honeypot, famous communication protocols (XMPP, MQTT, etc.) will get involved in, including the server and client part. By implementing both the server and client together, the honeypot is more flexible and could catch more data both from the server side and the client side.

Simulations of devices will also be done to attract the attackers. Related services like REST would be get involved in since most devices are using REST. This simulation will be based on the study of IoT devices on the market to make it looks real. One of the use case of the IoT device is the Philips Hue product, which is using REST in its bridge to communicate. More details will be explained later.

The idea of this honeypot is to make each components (communication services, devices simulation, etc.) work together. Then a simulation of an IoT platform is built up, which is very similar as the current real IoT platform. By simulating the same working scheme as the real platform, the problems that will be exposed by this honeypot would be more possible to show real problems. Thus reach the purpose of solving the research questions.

3.3.2. ThingPot Platform Model

ThingPot simulates both the IoT Platform and the IoT devices. Figure 3.1 shows the general idea of ThingPot that the relevant components are running on a simulation of IoT platform, including Frontend, Backend, IoT Devices Simulation, existing XMPP/MQTT services (servers, clients, libraries). All of these components compose the IoT platform which the hackers could get access to and interact with.

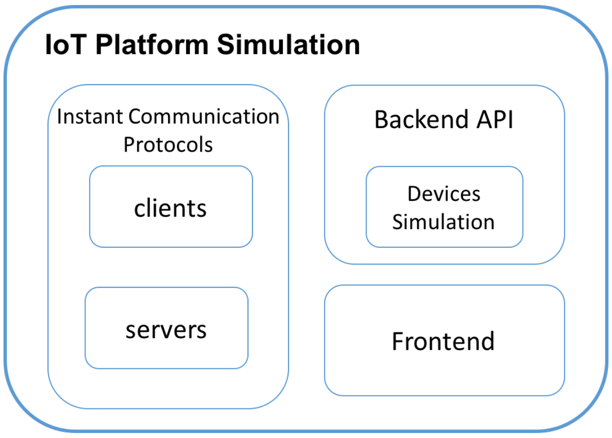


Figure 3.1: Idea of IoT Platform Simulation

- **Frontend** is a website that the user can see and interact with directly. The honeypot frontend shows the "information" of all the "devices" on the "platform" and the access to theses "devices". This makes the attacker believe that this is an IoT product and help the attacker to find the access to have further interaction.
- **Backend API)** could be built by REST API that could reply to the frontend's request. Or it could be simulation of IoT devices. The backend could be high interactive to the attacker that reply differently according to different requests. It would also learn from the attackers activities that log everything. After the analysis to the log the honeypot could be improved.
- **Devices Simulation** is a mechanism about how to simulate the IoT devices. To simulate the IoT devices, the study of the existing IoT devices is needed. By studying the real IoT devices, the real reply of the

devices will be known and thus do the simulation by making templates to reply.

- **clients of instant communication protocols** is an implementation of the client part of the protocols by existing libraries. The instant communication protocols that are often used in IoT is MQTT and XMPP. In a real IoT system, this could be the communicator between the devices, or between the users and devices that each device could be assigned with one account. By learning the mechanism of devices simulation, this could be another access for the attackers to reach.

- **servers of instant communication protocols** is an implementation of the server part of the protocols by the existing implementations or libraries. The server could be reached on the Internet, and communicate with other public servers. Thus the attacker could interact with the server.

Beside of these components that could be seen as a part of the simulation of an IoT platform. The ThingPot also contains another component that is the controller. By having the components cooperating together, the honeypot could simulate a effective and defective system for catching the malicious activities targeting to the IoT platforms on the Internet. However, it is not necessary that all the components are present at the same time to work. By choosing part of the components from the above, it would be enough for the current stage to study the attacker's activities and status on IoT devices and platforms. For an IoT Platform honeypot, the device simulation and instant communication protocols are necessary. Since the device simulation is one important feature of honeypot and it has the key influences on the performance of a honeypot. It could be a way that building up a honeypot for the Proof of Concept at the initial stage.

In this thesis, XMPP is chosen as the instant communication protocol in the ThingPot for further design. The components *Instant Communication Protocols, Backend API (with Device Simulation)* are chosen for further design. Further details could be seen in the next section.

3.3.3. ThingPot Components

In this section, more details about each component in the ThingPot Platform Model is explained.

XMPP Client and Server

Figure 3.2 shows the concept of the block diagram of the XMPP Client component. It contains the modules in different functions of an XMPP Client Honeypot:

- XMPP Main Module: Main module for controlling other modules of XMPP Client Honeypot.

- XMPP Connector Module (XCM): Connect to other external nodes.
  - Internal: Native components of the honeypot
  - External: Third-party components (e.g. libraries, tools, api, etc)

- XMPP Honeypot Module (XHM): Interaction part of this honeypot.
  - Handler:To handle the connector (as a binder) and the coming traffic (analyze and then call the Honeypot Management module)
  - Honeypot Management: To determine the interaction (service emulation) with the coming traffic

- XMPP Logging Module (XLM): In charge of logging of all events and honeypot data analysis.
  - Logging: To record all the honeypot events to generate the raw input
  - Honeypot Data Analyzer:
    - ◇ Classifier: To define the type of the input raw data (e.g. IP info, XMPP info, etc.)
    - ◇ Asset Parser: To parse any information that is useful and generate the output(e.g. json file)

- XMPP Honeypot Configuration (XHC): Modules to handle and manage the other modules.

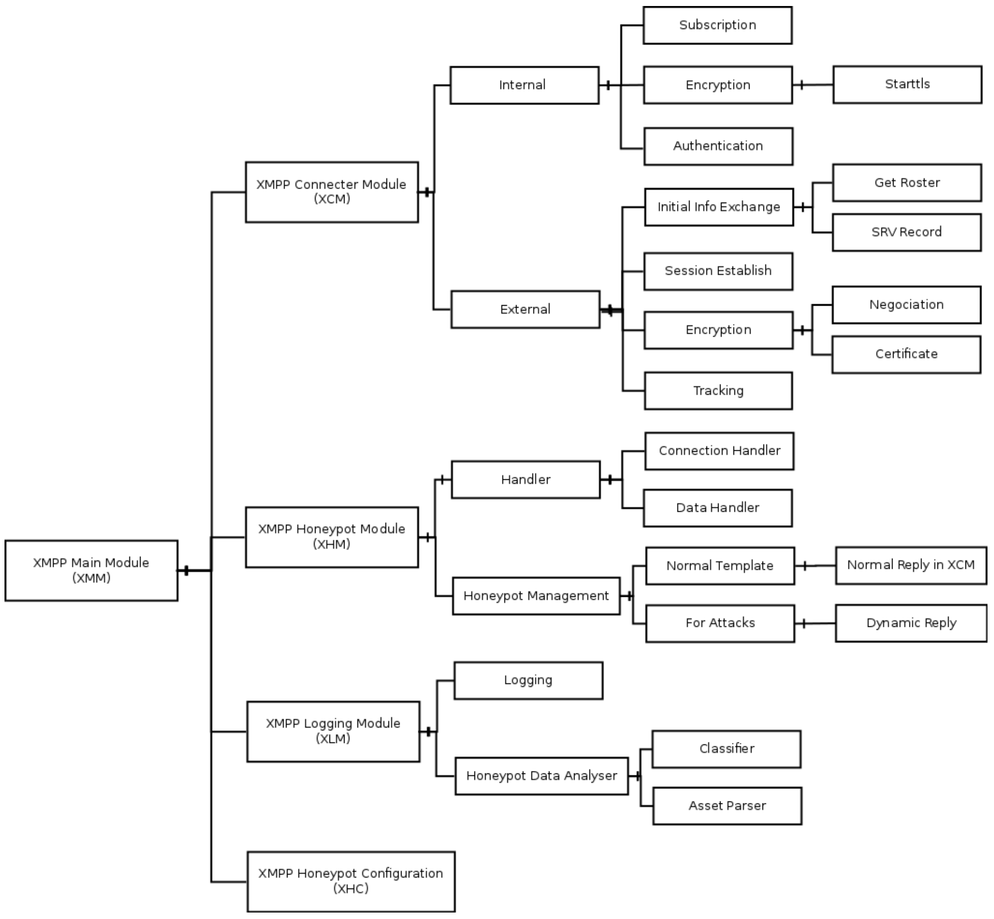


Figure 3.2: The Block Diagram of the XMPP Client Design

3.3.4. Attack paths

The workflow of the XMPP Client Honeypot can be explained by Unified Modeling Language (UML), which is a method to visualize the design of a system. In these UMLs, the modules that introduced in Figure 3.2 are involved. The UMLs show how and in what order that the modules work together to accomplish the functions of the honeypot when meeting different situations:

1. The attacker's activities starts from the client. Figure 3.3 is the UML that illustrates the workflow when the attacker attacks the system from the XMPP client. The presumed situation is that the attacker act as a XMPP client attacking the XMPP Client of the Honeypot.

2. The attacker acts as an unexpected, malicious server. The honeypot's XMPP client has been connected to a server, listening. And the attacker tries some activities like scanning. The honeypot can give a simple respond and take a record. Figure 3.4 shows the workflow of the honeypot for this situation.

3. The honeypot's XMPP client is connecting to a malicious server. A connected XMPP server, with a valid xmpp session and subscription. The presume situation is that the server is a malicious one that is trying to get data from the client honeypot. Figure 3.5 shows the workflow of the honeypot for this situation.

4. A malicious client connected to an external normal server. The presume situation is that the attacker act as a client connecting to the normal external server that can reach the Honeypot. Through this server the attacker try to get data from the client honeypot. Figure 3.6 shows the workflow of the honeypot for this situation.

The XMPP Server could be either a public one or a self-build one. By building up an XMPP server, we could gain more flexibility and data to analyze. But taking the complexity into consideration, the feasible way to build up an XMPP server is by using some existing implementations mentioned in Chapter 2. Public servers

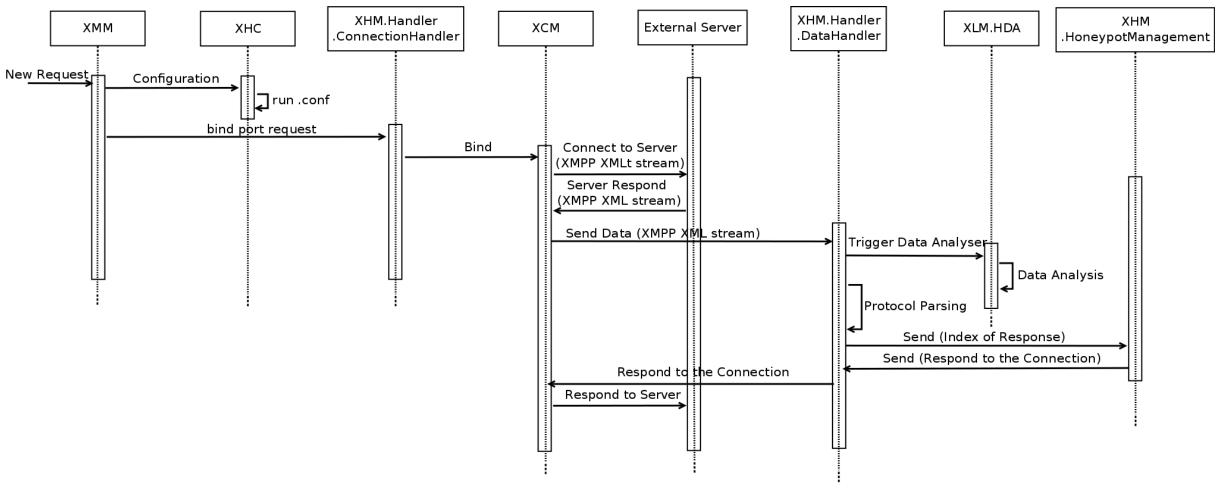


Figure 3.3: UML of XMPP Client Part of ThingPot, Start from Server

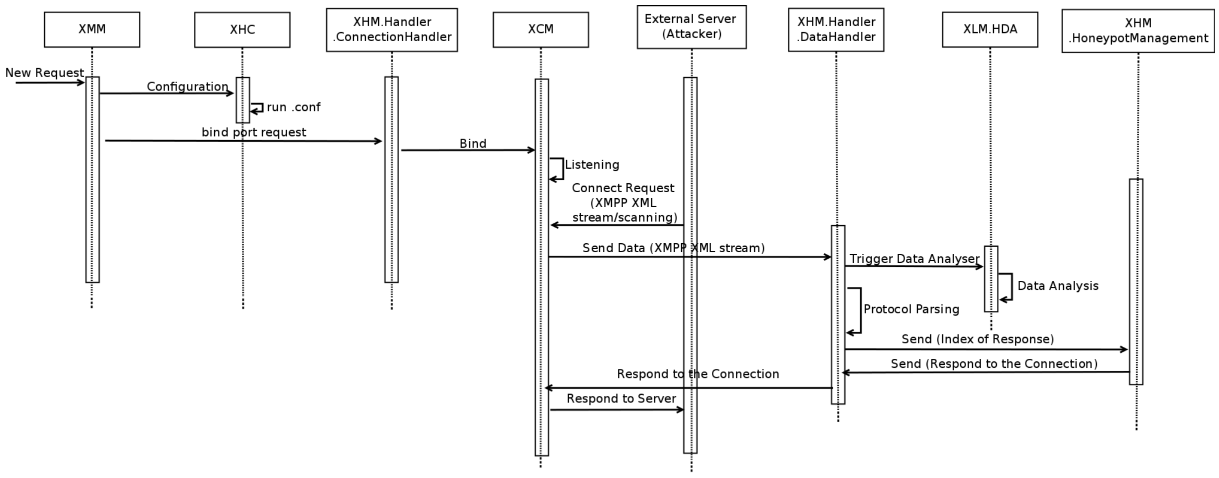


Figure 3.4: UML of XMPP Client Part of ThingPot, Start from Server

should be used, because it saves time and effort and could help a lot to get more information at the same time.

Backend API with Device Simulation

Many existing devices are using API as the interface for communication between smart devices. An API is going to be designed for ThingPot for the smart devices simulation. Four components are designed to work together in the API honeypot:

- API Accesses Handler: This part describes the accesses for the backend api that the attacker could reach the data or service. For example if it is a RESTful API built by Django, the access is the URL. The API Accesses Handler is the *urls* and *views* in the Django module that define the valid URL and to what service it could reach through the URL.

- API Response Handler: This part defines the how to respond to the incoming requests. It handles the request and call different data resource of the honeypot according to the request. This is based on the behavior of the real device that the honeypot is simulating. And the method of this Response Handler infects behavior of the honeypot and the ability of catching the attacker's activities.

- Data Resource: The data resource of the honeypot is a simulation of the real data on real device. By studying the behavior of simulated devices, the content of the data could be known. Thus the honeypot could build its "data resource". It will be called by the API Response Handler.

- Log System: Log System is important for the honeypot since it defines the way of data collection and store. The Log System should be designed in a proper way to collect and handle data properly.

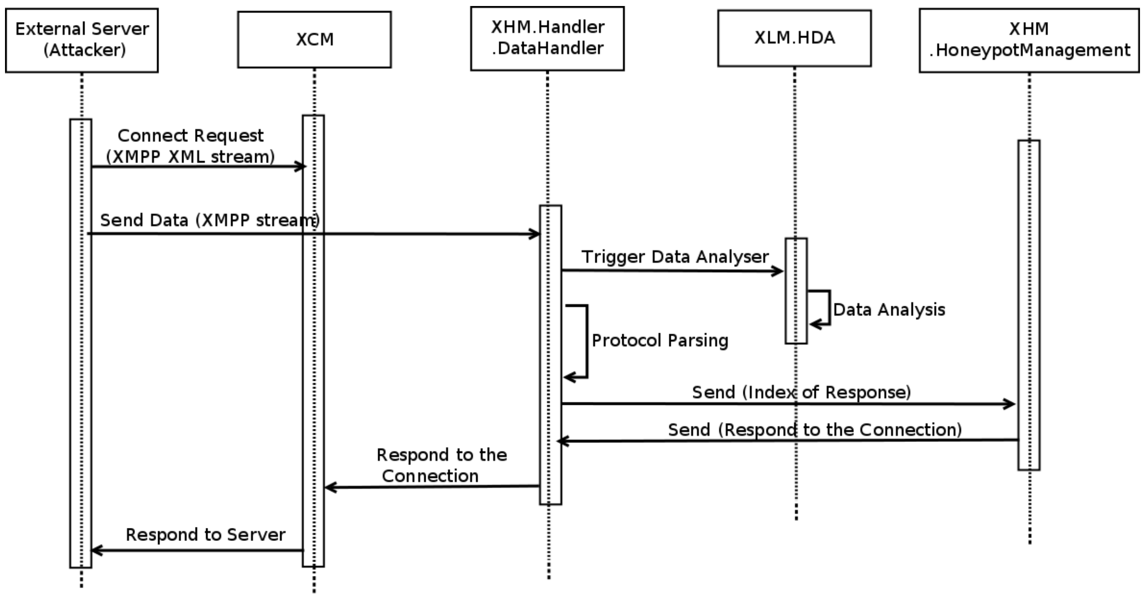


Figure 3.5: UML of XMPP Client Part of ThingPot, Start from Server

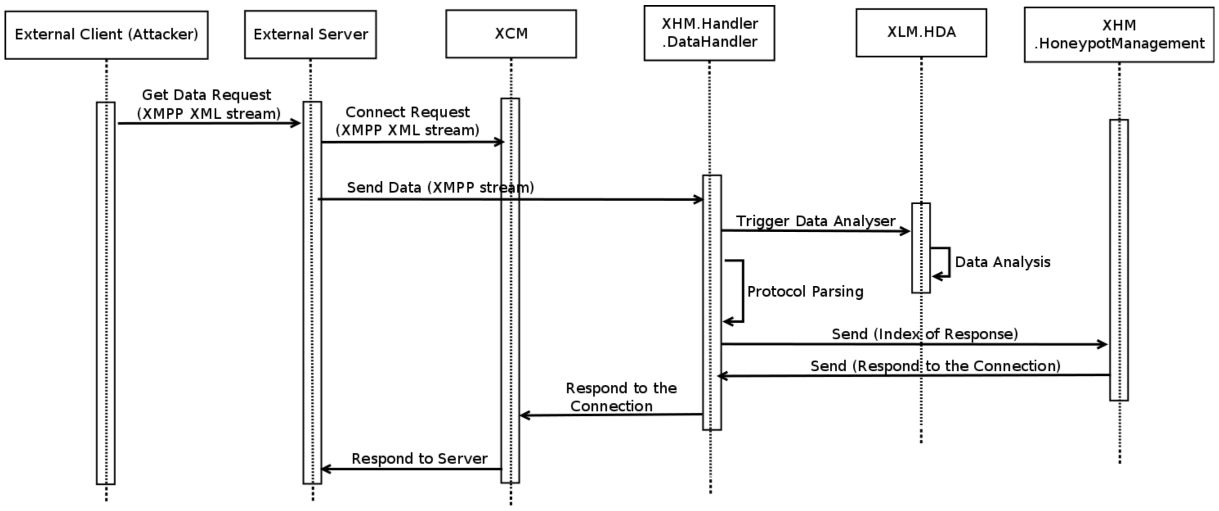


Figure 3.6: UML of XMPP Client Part of ThingPot, Start from Server

- Other Supporting Services: This module includes some services that is used for support the function of honeypot.

In summary, Honeypot is a technology that could be used for detecting the attacker's activities. It suits for the current situation of researching on IoT security problems. This chapter covered:

- ThingPot platform model is proposed in this thesis, giving a method of making an IoT honeypot in current situation of IoT platforms.
- Five main components were introduced. It is not necessary to pick all the components or implement all related protocols and services in order to deploy a Proof-Of-Concept (PoC)
- XMPP and REST are the chosen protocols and the XMPP clients, servers and REST API have been designed and are going to be implemented and described in detail in the next chapter.

4

ThingPot Implementation & Use Cases

The concept of ThingPot has been elaborated in chapter 3. A Proof of Concept of ThingPot will be implemented to see the behavior and find attackers activities. In this chapter, the Proof of Concept (PoC) as well as one of the Use Cases (Philips Hue platform) will be described in such a way can be demonstrated how ThingPot could be built, run and catch useful data.

4.1. Introduction

In this thesis, ThingPot will simulate the platform model described in Figure 2.3. The backend and instant communication protocols are going to be implemented. REST can be used to build a RESTful API. It is easy to implement and can provide the functions that a backend in an IoT Platform honeypot should have. Many literature points out XMPP as the IoT protocol for real time communication. Thus the XMPP and REST API are chosen to be used in the Proof of Concept (PoC) of ThingPot. Just as the description of the common IoT platform in section 2.3, by using XMPP and REST to work together can be enough to deploy an IoT platform that is similar to the existing implementations. Thus, Since the more exposed the honeypot is, the more chance for attackers to reach the honeypot, therefore, the services will be implemented in a way that it has as much visibility on Internet as possible. Figure 4.1 describes the physical topology of the honeypot implemented in this thesis. Details are explained in the following sections.

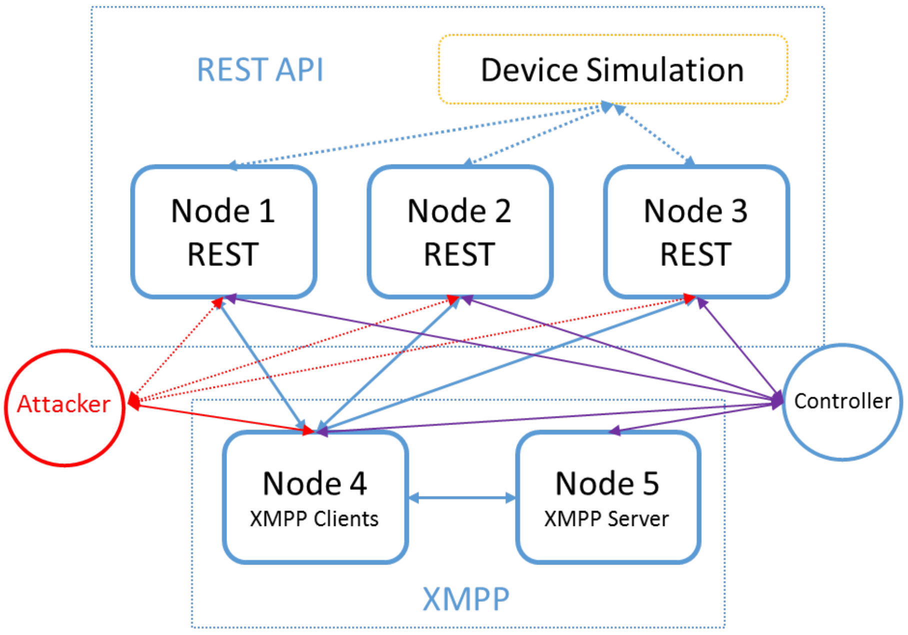


Figure 4.1: Physical Topology of ThingPot PoC Implementation

4.1.1. Network Topology

ThingPot implementation includes the three main components described in Figure 2.3. These components are distributed as follows:

- XMPP Nodes:
  Two nodes will be used as the XMPP part, including one node that run the XMPP client services and one node that run the XMPP server services.
- REST Nodes:
  Three nodes that have public IP addresses will be used to implement the REST part of the honeypot.
- Controller:
  In this implementation, there will also be a controller according to the design of ThingPot. The controller is just a computer that could locate anywhere with any IP address that could reach all the 5 nodes. The function of the controller is : logs observation, control, data backup, update of the code, etc. The controller has different accesses from the attacker by ssh in different ports.

Each node has been deployed in different devices including Raspberry Pi (RPi) and virtual servers running on the cloud (such as Amazon EC2 and Keyweb). Some nodes run an instance of the REST API backend, whereas others run XMPP services (server or client). The services are running within a container that is installed on the RPis since the container can be easily packed and moved to another node. Containers provide (to some extent) an additional layer of security, since network and process isolation from other services is in place [1][2]. In the context of Low interaction Honeypots, this layer of security can decrease the chances that, in a eventual actual compromise, the attacker reaches the actual system behind the service emulation of ThingPot. Moreover, on the node 1 and 2, proxies are running as a backup in any case API is temporarily down. The proxy could also help to mask the backend from the web scene. It is not necessary to have a proxy.

The REST part and the XMPP part could communicate each other by following specific rules that are the same as real systems. More details about the working mechanism of the nodes and communication between node will be explained later together with a user case. Table 4.1 shows the IP addresses and services on each node. Among this nodes, node 1 and 5 is actually running on one RPi but in different containers that can be reached by different ports. So these two services is totally independent and has no influence because of sharing one Rpi. Thus they are named in two numbers according to different services to make the topology clearer.

The left part (in red) of Figure 4.1 shows a potential attacker. The upper part (in yellow) shows the "device" which is behind the API. The red line shows the potential communication that the attacker could reach. Through a specific port, the attacker could reach the honeypot and interact with the honeypot. The port number that the attacker could reach and interact is shown in Table 4.1. And according to he XMPP characteristics, no public IP address is needed to make the XMPP client service reachable, because it connects to the XMPP server which has a public IP. The XMPP server will handle the traffic correctly. However, the

Node	Service	IP address	Port
1	REST	94.210.46.X	TCP/80
2	REST	83.84.11.X	TCP/80
3	REST	84.19.177.X, 84.19.176.X	TCP/80
4	XMPP client	internal server, no public address	-
5	XMPP server	94.210.46.X	TCP/5222 & TCP/5269

Table 4.1: Nodes' IP addresses list

attacker's target is assumed to be the "device". Thus there are two paths that the attacker could take to reach the "device". Figure 4.2 shows the attacker's paths.

4.2. Use Case Specification: Philips Hue

In order to proof the effectiveness of the design of ThingPot to get actual data related with IoT platforms, a use case of existing commercial implementation was chosen to be emulated: Philips Hue (Phue). The analysis

[1] https://docs.docker.com/engine/security/security/linux-kernel-capabilities
[2] https://www.oreilly.com/ideas/five-security-concerns-when-using-docker

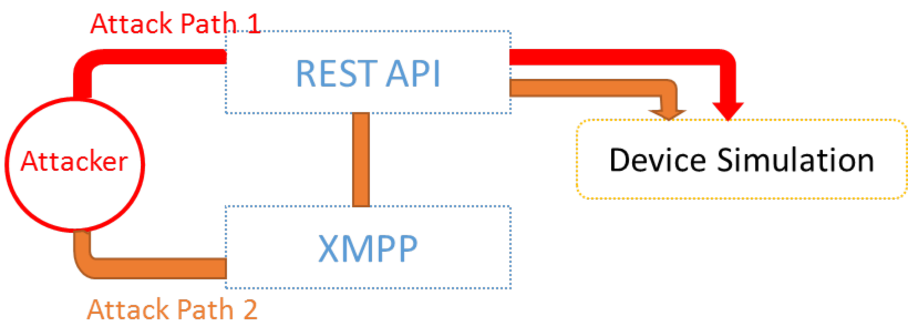


Figure 4.2: Attack Paths

and selection of a use case was based on the fact that it needed to be an existing model in which all the components of a general IoT platform were considered (i.e. devices, backend, communication protocol). Thus, Philips Hue was considered because it is a set of smart home devices that is one of the most popular smart wireless light bulb solutions that is using REST and it is easily available in the marketplace. In this thesis, the implementation of ThingPot will be based on the Phue use case, simulating platforms that integrates XMPP with Phue.

4.2.1. Philips Hue IoT Platform Introduction

The Phue product comprises of wireless LED light bulbs and a wireless bridge. The bulbs can be controlled using IOS and Android apps or through the hue website. Many people installed it at home and played with the app on smart phone and make their own lightening plan. Some of the developers find ways to integrate the Philips Hue with XMPP protocol. There are third-party implementation (source code) and technical manuals [72] of how XMPP works with Phue. Such IoT platform is being implemented by more and more people while the security problems are seldom considered. The problems come from either the communication protocols or during the integration of the devices and the protocols. Therefore, ThingPot is going to simulate the IoT platform that includes the Phue devices and the user's control panel (XMPP part).

Figure 4.3 shows the normal topology of the platform of integration of Phue and XMPP. 4 main parts are involved:

- XMPP Server: The XMPP server is for the XMPP communications. It transfers all the messages between the XMPP clients. This XMPP server can be a public server that is built by some one. Or you can also build your own XMPP server and configure it to communicate appropriately.
- Phue Devices: This part includes the Phue Bridge and the lamps (or other sensors like motions sensors). The Phue Bridge is a hub of the smart lamps and sensors. The lamps and sensors do not reach the Internet directly but through the Bridge. The Phue Bridge communicates with the lamps and sensors by ZigBee and communicates with the Phue server or users by REST API. It has all the information of the connected devices.
- PC-Integrator of Phue and XMPP: [72] introduced a way to integrate XMPP with Phue devices by running a provided code. By this integrator each smart device can have a JID. The user can send message to the JID to control the light like turn on or off, adjust the brightness, etc. This integrator is a script that use sleekxmpp and Phue library. You can also change the code to fit self requirement. Basically, it receives the xmpp message, then translate it into a REST request and send the request to the Phue bridge. The code can be run on a device that is in the same network with the bridge.
- XMPP client: User can apply for an XMPP account on any XMPP server that can be reached. And then communicate with the Phue Devices by XMPP.

4.2.2. Resource Data Structure

In this thesis, the Philips Hue White Starter Kit is used for studying the behavior of Phue IoT Platforms. Based on observation on real devices on the developer forum of Phue. The data of Phue Bridge is using Json format

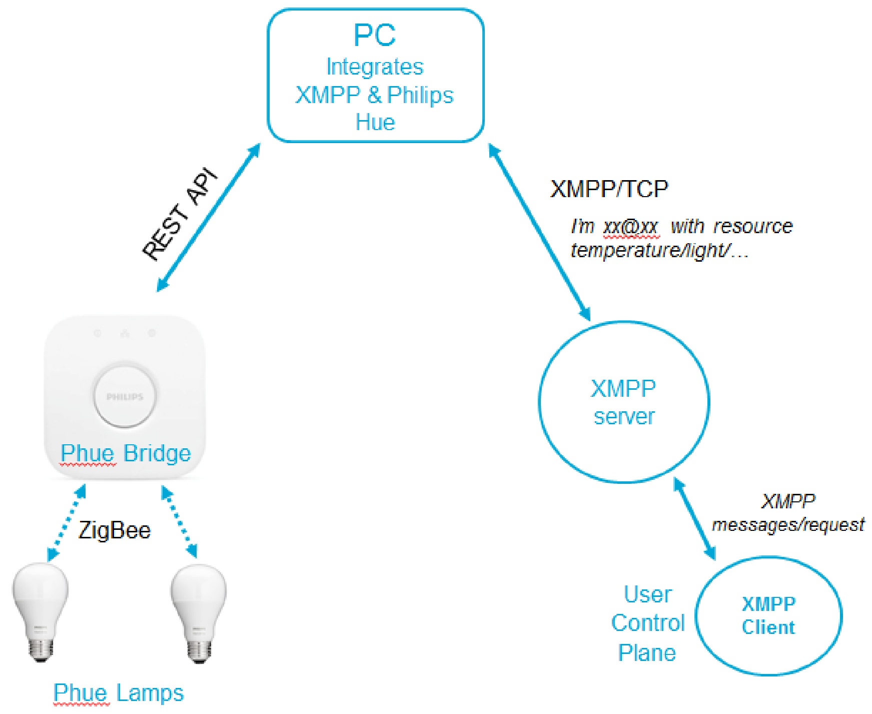


Figure 4.3: Philips Hue & XMPP Integration

to store and exchange data. Figure 4.4 shows the data structure of Phue Bridge. It includes all the important information of this bridge and is in a layered structure that it has the objects and subitems under these objects that is related. We can call the objects on the first line the first layer and the subitems are the second layer. There are also subitems of the second layer, which can be see as the third layer. Under the object *light*, each connected light has a number and the information of this light is included. Figure 4.5 shows the subitems of object *lights*. From this we can see that the status of the light and the useful information can be found. And the data of course is dynamic that changed according to the real status of the bulb.

4.2.3. Communication Mechanism of the Bridge

According with a manual [54] for Phue developers, the Phue Bridge has an API to exchange information with the user. It is a RESTful interface over HTTP that every light and every controllable parameter of the lights has a valid URL. Thus the developer and user could simply send the new value to a specific URL to control the device (via HTTP PUT or POST). The data could get by the developer or user by sending a HTTP GET request to the URL. The URL always starts with *http://<bridge IP address>/api* and the authentication mechanism requires a valid username that is contained in the *config-whitelist* for most of the commands. So most the the URL that is functional starts with *http://<bridge IP address>/api/<username>*. Then the link for different resources is just add the name of the object that is shown in Figure 4.4. For example, to get the information of light1, the URL is *http://<bridge IP address>/api/<username>/lights/1/*. Bye sending a HTTP GET request to this URL with a valid username, the bridge will send back a Json data as Figure 4.5 shows with contents included. Details could be found in the hue developer website [54].

4.2.4. Control the lights

The controllable parameter for the Philips Hue White is:

- *bri*: Integer. The brightness of the light, ranging from 0-256
- on: Boolean. The value is *True* or *False* that indicated the bulb is on or off respectively.

There are more controllable parameters on the colourful bulb like *hue, sat, colormode, xy, ct, etc.*. Since the simulation is about the Philips Hue White, these parameters are not taken in to consideration.

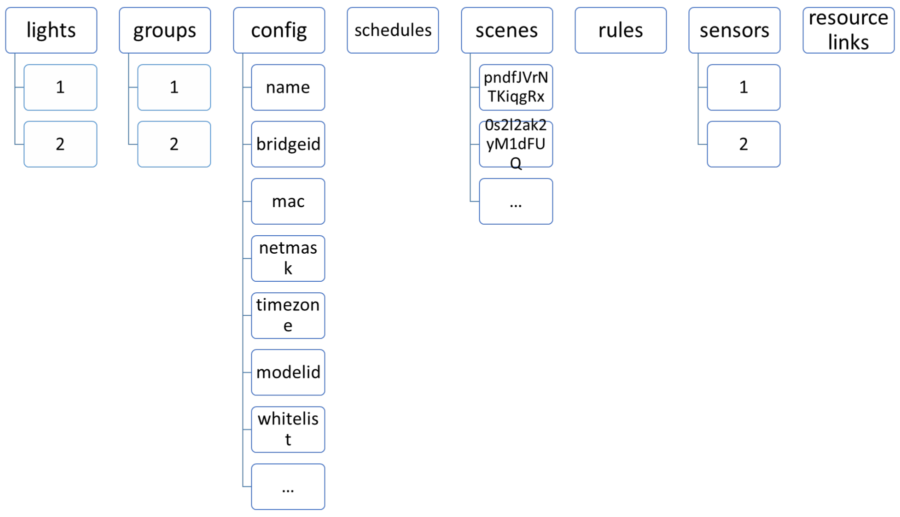


Figure 4.4: Resource Data Structure of Phue Bridge

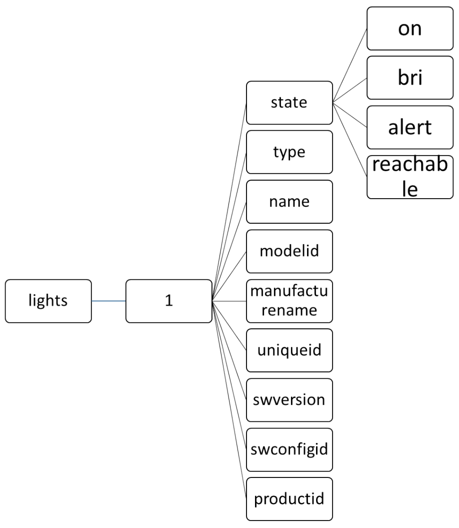


Figure 4.5: Structure of Phue Bridge's data object *lights*

To control the smart bulb (change the light's state), we can send a HTTP PUT to the bridge's API. The URL is *http://<bridge IP address>/api/<username>/lights/1/state*, with a body in Json format: *"on": true, "bri": 200*. If it is successful, a successful message will be sent back and the bulb's status will be changed.

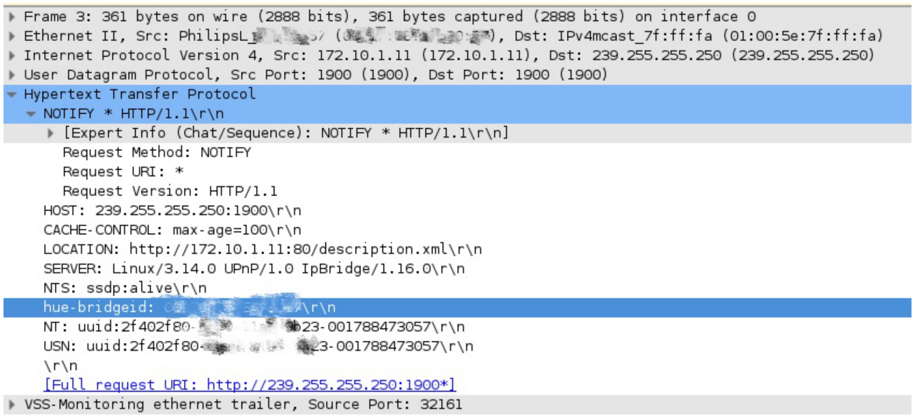
4.2.5. Security analysis of Phue IoT Platform for Honeypot implementation

The Phue system is a mix of network protocols and application interfaces. The original plan of Phue devices is that users can control the lights by using the app on a mobile phone or hue website if they have an authorized Philips account. But because of the limitation of the app or website (you have to install the app on your phone, you can only use this for Phue system), developers would like to integrate the XMPP with Phue to make it more flexible and can cooperate with other smart devices. In this thesis, we call this integrated platform the Phue (Philips Hue) IoT Platform. This brings convenience but also danger. Through the study of Phue products, vulnerabilities of Phue have been found:

1. [23] lists several threats that Philips hue is facing.
   It explains how the hue bridge uses the whitelist for authentication and how the vulnerabilities could be made use of. The threats includes perpetual blackout, potential for remotely issued blackouts based on password leaks and poisonous recipes and platform compromise.

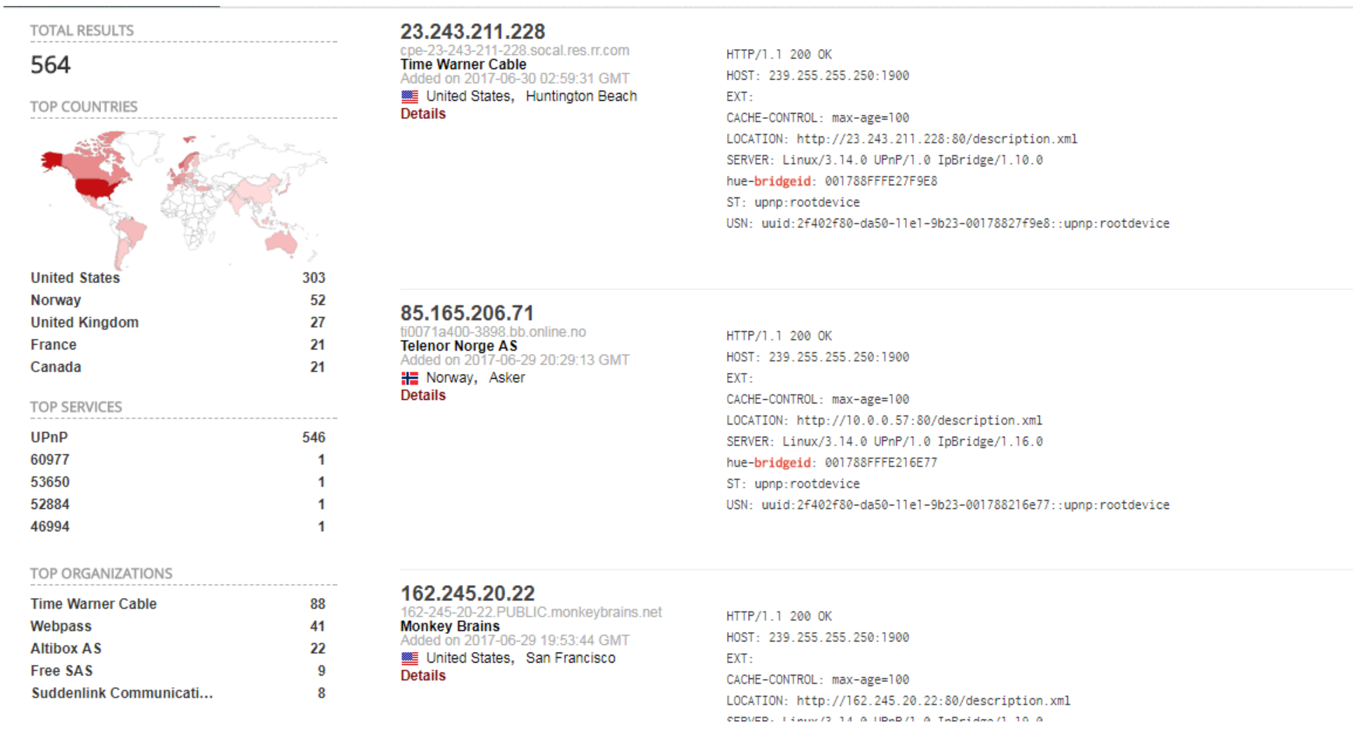
2. The location of a bridge on a network can be found in several ways:

(a) Information is exposed because of the Simple Service Discovery Protocol (SSDP) notify packages. The hue bridge is using UPnP for discovering. It broadcasts its presence frequently by SSDP. Figure 4.6 is the package caught by Wireshark that is sent by a hue bridge. It is in plain text and can be seen on Shodan.io, a search engine for the Internet-connected things. This exposes the bridge's "LOCATION" which includes the internal location of the bridge (172.10.1.11) and the bridge id. On Shodan.io, a number of hue bridge can be found, if we search with the keyword "bridgeid". From Figure 4.7 we can see the public IP addresses of the hue bridges. And the internal location is included in the item "LOCATION". When the internal location and public IP are the same, the bridge can be reached by the public IP.



```
▸ Frame 3: 361 bytes on wire (2888 bits), 361 bytes captured (2888 bits) on interface 0
▸ Ethernet II, Src: PhilipsL_[redacted] ([redacted]), Dst: IPv4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
▸ Internet Protocol Version 4, Src: 172.10.1.11 (172.10.1.11), Dst: 239.255.255.250 (239.255.255.250)
▸ User Datagram Protocol, Src Port: 1900 (1900), Dst Port: 1900 (1900)
▾ Hypertext Transfer Protocol
  ▾ NOTIFY * HTTP/1.1\r\n
    ▸ [Expert Info (Chat/Sequence): NOTIFY * HTTP/1.1\r\n]
      Request Method: NOTIFY
      Request URI: *
      Request Version: HTTP/1.1
    HOST: 239.255.255.250:1900\r\n
    CACHE-CONTROL: max-age=100\r\n
    LOCATION: http://172.10.1.11:80/description.xml\r\n
    SERVER: Linux/3.14.0 UPnP/1.0 IpBridge/1.16.0\r\n
    NTS: ssdp:alive\r\n
    hue-bridgeid: [redacted]\r\n
    NT: uuid:2f402f80-[redacted]b23-001788473057\r\n
    USN: uuid:2f402f80-[redacted]3-001788473057\r\n
    \r\n
    [Full request URI: http://239.255.255.250:1900*]
▸ VSS-Monitoring ethernet trailer, Source Port: 32161
```

Figure 4.6: SSDP package sent by hue bridge



TOTAL RESULTS

564

TOP COUNTRIES

United States	303
Norway	52
United Kingdom	27
France	21
Canada	21

TOP SERVICES

UPnP	546
60977	1
53650	1
52884	1
46994	1

TOP ORGANIZATIONS

Time Warner Cable	88
Webpass	41
Altibox AS	22
Free SAS	9
Suddenlink Communicati...	8

23.243.211.228
cpe-23-243-211-228.socal.res.rr.com
Time Warner Cable
Added on 2017-06-30 02:59:31 GMT
United States, Huntington Beach
Details

```
HTTP/1.1 200 OK
HOST: 239.255.255.250:1900
EXT:
CACHE-CONTROL: max-age=100
LOCATION: http://23.243.211.228:80/description.xml
SERVER: Linux/3.14.0 UPnP/1.0 IpBridge/1.10.0
hue-bridgeid: 001788FFFE27F9E8
ST: upnp:rootdevice
USN: uuid:2f402f80-da50-11e1-9b23-00178827f9e8::upnp:rootdevice
```

85.165.206.71
ti0071a400-3898.bb.online.no
Telenor Norge AS
Added on 2017-06-29 20:29:13 GMT
Norway, Asker
Details

```
HTTP/1.1 200 OK
HOST: 239.255.255.250:1900
EXT:
CACHE-CONTROL: max-age=100
LOCATION: http://10.0.0.57:80/description.xml
SERVER: Linux/3.14.0 UPnP/1.0 IpBridge/1.16.0
hue-bridgeid: 001788FFFE216E77
ST: upnp:rootdevice
USN: uuid:2f402f80-da50-11e1-9b23-001788216e77::upnp:rootdevice
```

162.245.20.22
162-245-20-22.PUBLIC.monkeybrains.net
Monkey Brains
Added on 2017-06-29 19:53:44 GMT
United States, San Francisco
Details

```
HTTP/1.1 200 OK
HOST: 239.255.255.250:1900
EXT:
CACHE-CONTROL: max-age=100
LOCATION: http://162.245.20.22:80/description.xml
SERVER: Linux/3.14.0 UPnP/1.0 IpBridge/1.10.0
```

Figure 4.7: Screenshot of *shodan.io* by Searching "bridgeid"

(b) [65] provides a way to find a phue bridge on the network by using nmap and arp lookup. Then a hue bridges IP and MAC address can be found.

(c) When is using the same public IP with the bridge, one can send a GET request to the API URL *meethue.com/api/nupnp* to get the internal location (private IP address) and bridgeid of the bridge.

3. Because of 2, one hue bridge's IP and MAC address is found and can be reached by the attacker. By learning the scheme of the hue bridge, the attackers could make the hue bridge face potential attacks:

   (a) More information (swversion, modelid, apiversion, etc.) can be exposed. By requesting */api/config* or */api/ANY_STRING/config* to the bridge's IP, the bridge will reply a Json file including those information without any authentication.

   (b) According to the Hue Developer Program[54], a username can be created by first press the button on the bridge, then send a post request to the bridge with particular body content. With this username, which will be stored on the whitelist of the bridge, one can have the permission of controlling and configuring all the smart devices under this bridge. So when the bridge can be reached by the attacker, the attacker can send this request continuously to the bridge. The attacker has the chance to get a valid username when the user of the bridge press the button by accident or reconfigure the bridge.

4.2.6. Integration with XMPP

[72] is the tutorial to link a JID with a Phue light. It provides the python library [67] for the Phue system. In this thesis, the library is named as *phue-lib*. Then the light can be controlled by communicating with the JID through XEP0323 read or XEP0325 control or a simple chat. To achieve this, a list of things should be prepared:

- An environment that could run python scripts.
- A JID for each bulbs. The JID could be applied on a lot of public XMPP servers. The developer could also build a private XMPP server and register accounts (JID) on it. One JID for the developer to control the bulbs.
- Reachable IP address and port number of the Phue bridge.
- Library Sleekxmpp (XMPP client library) for building XMPP. communication

The phue-lib is a connector and translator between the XMPP connection and the API of the bridge. For the XMPP part, it connects to the XMPP server that the JID belongs to, telling the server that the JID is "online" and is listening. This is achieved by the Sleekxmpp library. For the API part, it defines the commands that could be sent through the chat between two JIDs (the bulb's JID and the developer's JID) and the commands will be translated into HTTP requests and sent to the Phue bridge. Moreover, instead of chatting, it could also be communicated with through XEP0323 and XEP0325, which is XEPs proposed specifically for IoT devices read and control. Phue-lib parses the traffic sent through XMPP stream and then generates HTTP request accordingly.

In this way, one JID that could be reach by another JID is bound with one Phue smart bulb through Phue-lib and the Phue bridge.

4.3. Proof of Concept of ThingPot

In this section, the actually environment and settings of the honeypot implementation will be introduced. The implementation is based on the research in preceding chapters and the study on Philips Hue devices. The physical Topology of ThingPot PoC is shown in Figure 4.1. Phue devices and Phue platform simulation will be done in this honeypot.

4.3.1. Simulated Scenario

One feature of the honeypot is "baiting", which means the honeypot should convince the attacker to believe that this is a system that is worth to attack. Thus the more similar the honeypot is going to be to a real scenario, the more likely that the honeypot is going to detect and log attackers' activities. The honeypot is simulating a scenario that is common in real life, this make it more convincing and attractive to the attacker to attack. In this section, a scenario that the honeypot is going to simulate and present to the attackers is going to be described.

The simulated scenario is: A Philips Hue system that has two smart bulbs connecting to one Phue bridge, locating somewhere in a building. And the Phue bridge API has a public IP address that could be reached on the Internet. The user has developed an XMPP client could communicate with the server. A JID has been registered on an XMPP server for controlling and monitoring one smart bulb. One JID binds one smart bulb that the user could control the bulb that the JID represents via an XMPP communication to the JID.

In fact, there are a lot of JIDs running in order to increase the possibility of being attacked. Figure 4.8 shows the relationship between the JIDs that is listening on the XMPP client honeypot and the "smart bulb" available on the REST honeypot. The solid line means that the JID is linked to the "smart bulb".

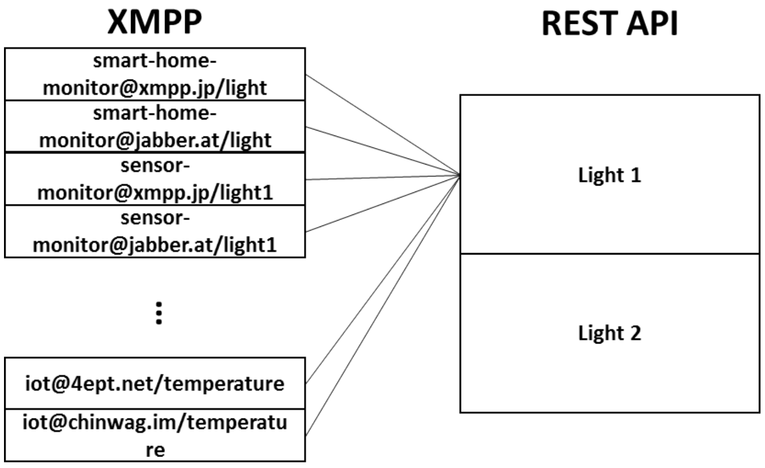

XMPP
REST API
smart-home-monitor@xmpp.jp/light
smart-home-monitor@jabber.at/light
sensor-monitor@xmpp.jp/light1
sensor-monitor@jabber.at/light1
iot@4ept.net/temperature
iot@chinwag.im/temperature
Light 1
Light 2

Figure 4.8: The account and "device" relationship between XMPP and REST honeypot

4.3.2. REST Implementation

The ThingPot PoC contains a RESTful API that simulates the behavior of the Phue bridge API. Django is used to build an API. Figure 4.9 is the block diagram of the REST honeypot in the real implementation. It follows the design of Backend API described in section 3.3.3. It is designed for simulating the behavior of Phue bridge with some features to make the attacker's interaction easier and deeper. From the block diagram we can see that the honeypot REST is mainly in two parts, the *api* and manager. This is structure is based on the Django design, where the *manager* is for settings and management of the api and the *api* is for function realization of the api. Under the item *api*, there are mainly 6 components for the honeypot REST:

- *urls*: This is the configuration of the URL patterns of the honeypot. It maps URL patterns (simple regular expressions) to Python functions (views).
- *views*: This is the python code to take requests and return responses.
- *Data Resource*: Here include three Json files that is a template of response. *template* and *config* is following the format of Phue bridge resource data structure with contents that looks like real device. *tempfile* is an additional feature of the honeypot implementation that the real Phue bridge does not have. It is designed that the attacker could get to know more information of the "device". Thus go for further interaction and the honeypot could catch more activities and data.
- *GetTemplate*: This is a class that can be called by *views*. It is a handler to handle the response to the income requests, managing what to respond and call the correct data from the *Data Resource*.
- *Service*: *Service* is a class that includes the functions that will be called for some specific functions that is not directly related to the honeypot, such as generating a random string, parsing the HTTP header, etc.
- *middlewarelogrequest*: Django is using the "middleware" mechanism for the "plugin" services. Here a customized middleware is built and placed under the *api*. This is a service for the logging system of this honeypot implementation and export it into a .json file. It parses all the incoming requests and log

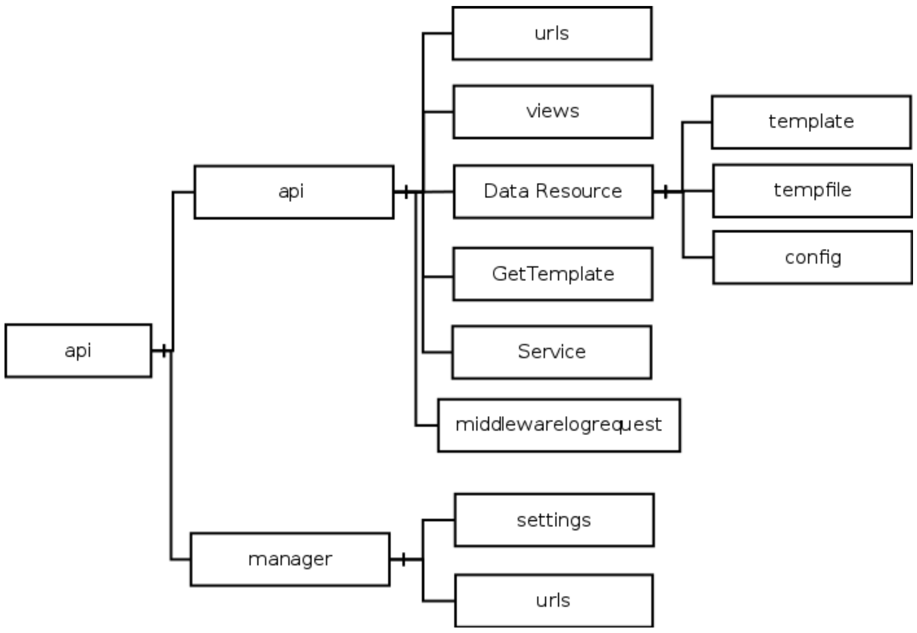


Figure 4.9: Block Diagram of the Honeypot REST implementation

down useful information in Json format, which is a flexible data format that is easy for further analysis. It is a very important component in this implementation because logging down the information in a proper way is essential for the data analysis of the honeypot data. It means that the there should be no missing of useful information in the log, and the data format should be clear enough to be analyzed in a easy way. Section 4.5 explains details of the log system, including the REST and XMPP part and how these two log systems cooperate.

The URL definition and data resource management of the ThingPot implementation follows the basic rule of Phue bridge API. But the actions are not exactly the same as real one. The honeypot API simulates some of the actions of the Phue bridge but not completely. Simulated actions includes:

- Registration for a new username:
  A new username that will be included in a whitelist can be registered by sending a POST request to the bridge after the button of the bridge is pressed (in 30 seconds). It will return a success message includes a 40-digit string, which is the new username if succeed. Otherwise it will send an error message. The honeypot is simulating this behavior but accepting all the request and return a success message with a 40-digit string generated randomly. This is designed to make the attacker believe they have successfully get a valid username to have the privilege to go further. Thus the attacker will have more actions instead of just trying to register a new username.

- Get all information of this bridge:
  As mentioned before, the real Phue bridge's data is in layered structure and is in Json format. The complete Json data could be get by an HTTP GET request */api/<username>* where *<username>* should be valid. The honeypot

- Get the sub layer information:
  "lights"/"config"/"scenes"/"whitelist"/etc. Any information in any layers can be get by an HTTP GET request with a valid URL. For example, to get the "whitelist" of the bridge, by checking the the URL is *http://<bridge IP address>/api/<username>/config/whitelist.*

- Control the light:
  The value of the controllable parameters mentioned before can be "controlled" by the HTTP request in this honeypot API. When the attacker send a "valid" HTTP PUT request, the API will reply a "success" message in the same format as a real Phue bridge.

- Change the other parameter by an HTTP PUT:
  This is similar to "Control the light". On the real Phue bridge, some of the parameters (e.g. item *name* under *config*) could be changed though the PUT requests. The honeypot simulates this behaviour that it would send a success message back when receiving such a request.

4.4. XMPP Implementation

The implementation of the XMPP part of the honeypot follows the design that is shown in Figure 3.2 and the workflow that is shown in the UMLs (Figure 3.3, 3.4, 3.5, 3.6). Figure 4.10 shows the the components (main design in in Figure 3.2) that were implemented for the proof of concept. The highlighted part (in white) are the ones that are implemented. It can both connect to a real Phue bridge as an independent honeypot and

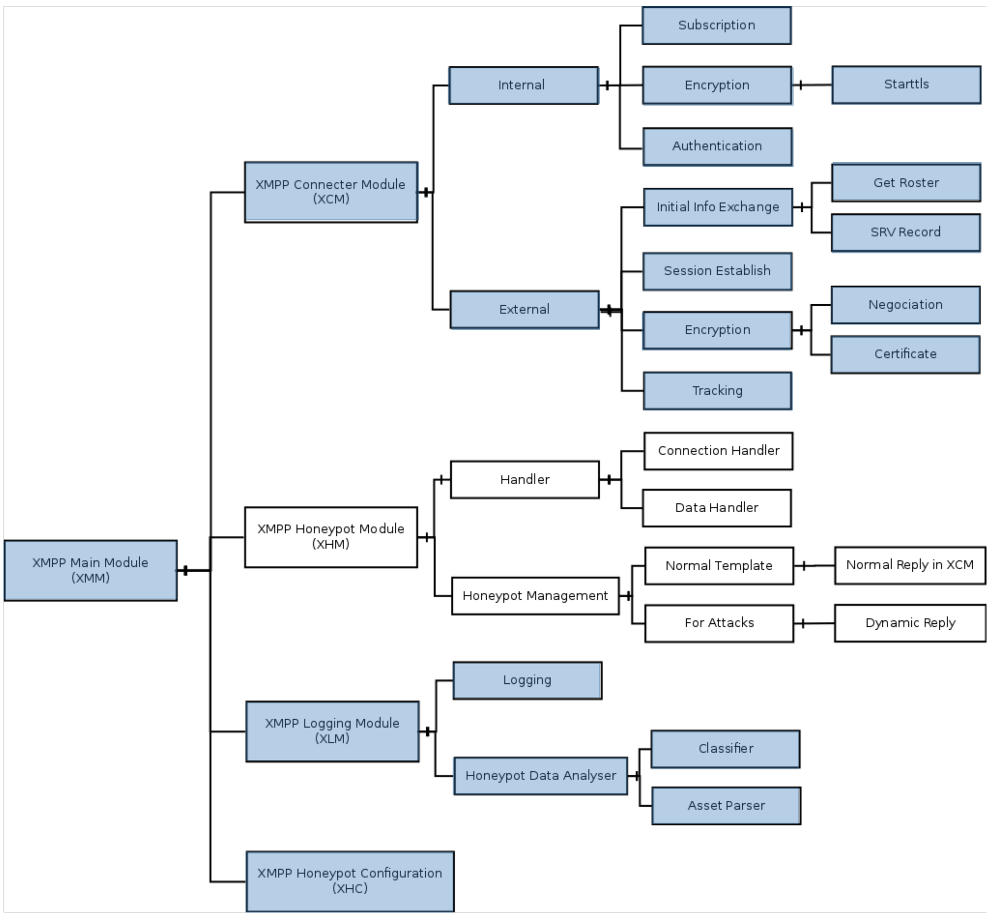


Figure 4.10: Implemented modules of XMPP client honeypot

connect to the api of the honeypot. It includes XMM, XCM, XHM, XLM, XHC:

- *XCM*: This is the connector module for the XMPP Honeypot. Functions of connecting with the XMPP server are included. The script *IoT_PhilipsHueApi_meng_api.py* is going to be called by the XMM in the Philips Hue use case. It tells the right XMPP server that one JID (as one of the input parameter) is online. And one JID is linked to one device on the Phue bridge. This script defines the commands that could be sent through XMPP chat or through XEP0323 and XEP0325. It also translates these commands into valid requests to the "Phue bridge". In this implementation, the "Phue bridge" is the REST part of the honeypot, which is an api built by Django. So the XCM here is a connector to the XMPP server and the api. It is built based on *sleekxmpp* and *phue-lib* by importing libraries and overwriting some of the functions. In this implementation, the XMPP client honeypot connects to the REST API of the honeypot. And all the JIDs are registered for the "light 1" of the "Phue bridge" in the REST honeypot.

- *XLM*: This part is for handling the incoming requests and system behavior into logs. In the real implementation, the XLM is placed inside the XCM. It defines a new logger that is going to log some of the message in Json format and export into a file. The way of logging is very important for the XMPP Honeypot. It should be complete and clear for further data analysis. Details about hte logging method will be explained in section 4.5.

- *XHC*: Here the configuration file is placed. The configuration file of the Honeypot is in YAML (Yet Another Markup Language) format. YAML is seen as one of the main markup language for Python configuration. It has 6 main components:
  - *names, domains, resources*: These three items are for the JID. Since the script in XCM can only register one JID to the server for one "device".
    To increase the possibility of being hacker or scanned, the JID listening on XMPP servers should be as much as possible. So the *names, domain* and *resources* here refer to the node name, domain and resource name in a JID respectively. The combination of the three items would be a JID. In the configuration file, a list of node name, domain, and resources will be put under each item. Then when the configuration file is called by XMM, there will be an iteration of the combinations of these three items. The condition is that the combination which build a JID should has been applied on the XMPP server. In this implementation, a number of JIDs that has the same node name in different XMPP servers (which makes the domain different) have been applied. And the JIDs that are used in this honeypot is named in a way that obviously it is for IoT devices. For example the node names are "smart-home-monitor", "sensormonitor", "iot", "home", etc. Thus it could be more attractive for the attacker. Table 4.2 is a table that shows the JID I have registered. The first column is the node name of a JID. The first row is the domain name (XMPP server) of a JID. A tick in the crossed cell means that this combination *(node@domain)* has been applied succeffully and could be used in the XMPP honeypot. A cross in the crossed cell means that this JID has been occupied by someone else and is not possible to be used in the XMPP honeypot. The blank cell is that this JID has not been tried, not knowing if it is available or not. This table is a reference on what a JID used by the honeypot would be like.
  - *apiip*: The ip address of the "Phue bridge"
  - *apiusername*: The 40-digit username for authentication of "Phue Bridge".
  - *filepath*: Path to store the log file
- *XMM*: A script called *scan.py* is placed here, which can be seen as the main function. It will call the configuration file, the XCM part in a organized way.

Another feature according to the design of the XMPP client honeypot mentioned before is one JID (named as JID manager) for controlling and monitoring the other JIDs. This is realized in this implementation which makes the developer monitor the status of all the JIDs in a intuitive way. When the JID is once online, it will send an XMPP message to the JID manager saying "XXX(the JID) is online". If the developer log in to the JID manager account, it is intuitive that which account is listening successfully.

4.5. Log System

The log system of the honeypot implementation is essential since the data analysis is mainly based on the logs generated by the log system. Each part (REST and XMPP) of the honeypot has its own log system. In the mean time, the log systems also shared some common features for correlation, which would contribute to analysis.

4.5.1. The REST Log System

The REST log system classifies the logs into two classes: normal Django console log, and the incoming requests and related response. Different kinds of logs will be exported into different files.

The log system has its own way of naming the log files. In order to make it clear and avoid being overwritten by reruning the honeypot, timestamp is used in the name of the log files. Thus the name of the log file will never be the same each time when the honeypot is run. Moreover, to distinguish different classes of logs, the names of the log files are started with different names. The normal Django console log is started with *sys* and the name is *"sys + TIMESTAMP.log"*. The requests log is started with *json* and the name is *"json+TIMESTAMP.log"*

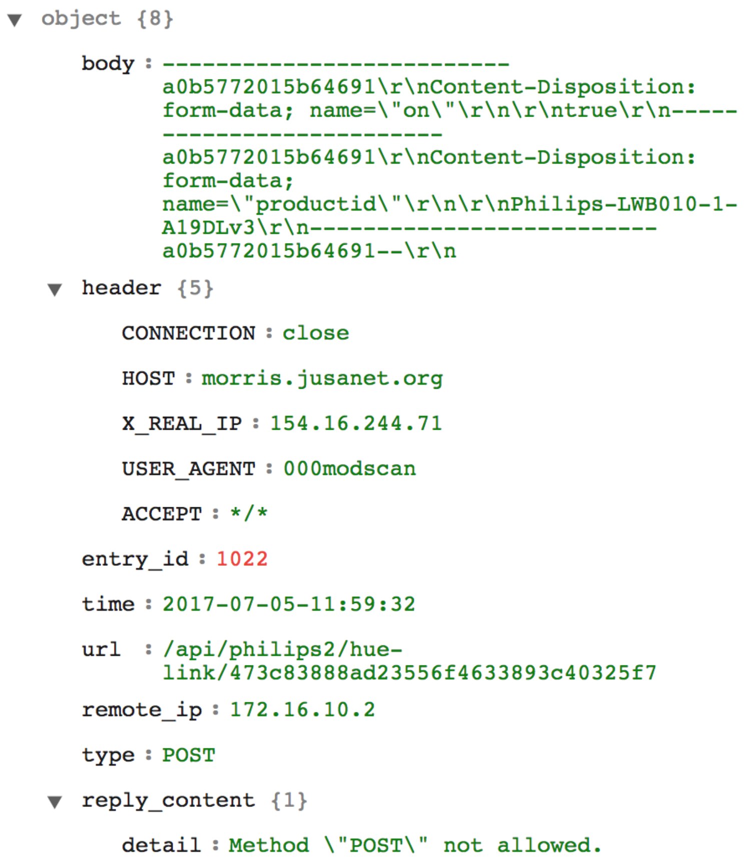
	xmpp.jp	jabber .at	4ept .net	chatme .im	jabber. net-work	chin wag.im	movim .eu	morris.ju sanet.org
smart-home-monitor	✓	✓	✓			✓		✓
sensor-monitor	✓	✓	✓			✓		✓
smarthom emonitor	✓	✓	✓	✓		✓		✓
phuebridge	✓		✓			✓		✓
lightsroup	✓		✓			✓		✓
lightgroup	✓		✓					✓
sensorsgroup	✓		✓					✓
sensrogroup	✓		✓					✓
switches	✓		✓			✓	✓	✓
wemoswitch	✓		✓					✓
wemo-switch	✓		✓					✓
iot	✓	✓	✓			✓		✓
lights	✗	✓	✓	✓	✓			✓
light	✗	✓	✓			✓		✓
monitor	✗	✗	✓	✓	✓	✓	✓	✓
switch	✗	✓	✗	✓	✓	✓		✓
home		✓	✓	✓	✓			✓

Table 4.2: Applied JID used for the honeypot

The logs of Django console is done by the settings of Django. It logs down all the system activities of Django.

The requests log system is logging every requests and corresponding response no matter it is valid or invalid. This is done by making a middleware of Django to handle the incoming requests first. After parsing the header and other useful information, a log message is generated in Json format and saved in a specific file. Each line of this file is one entry of Json for one request. Figure 4.11 shows an example from the real log that logged by the REST honeypot. It is one entry of the log file and is parsed by an online json parser JSON Editor Online [3]. The keys of this Json entry are:

- time: The current time. Timezone might need to be set manually.
- body: The body of the request. When the request is a PUT or POST, the body normally is not empty. The attackers might send some strange commands or exploit through the body of the request.
- header: The header of the HTTP request contains a lot of useful information such as the user agent, IP address, the host name, etc.
- url: The URL of the request.
- entry_id: The counter of the entries. This is unique in one log file and can be used for giving an id for each entry of Json log.
- type: The type of the HTTP request, such as "PUT", "POST", "GET", etc. This is an important value that could be analyzed later.
- remote_ip: The remote ip of the attacker.
- reply_content: The reply of the honeypot according to this request will be logged here. And according to the REST implementation, the response is always in Json format by simulating the real Phue bridge.



```
▼ object {8}
    body : --------------------------
           a0b5772015b64691\r\nContent-Disposition:
           form-data; name=\"on\"\r\n\r\ntrue\r\n-----
           ---------------------
           a0b5772015b64691\r\nContent-Disposition:
           form-data;
           name=\"productid\"\r\n\r\nPhilips-LWB010-1-
           A19DLv3\r\n--------------------------
           a0b5772015b64691--\r\n
  ▼ header {5}
        CONNECTION : close
        HOST : morris.jusanet.org
        X_REAL_IP : 154.16.244.71
        USER_AGENT : 000modscan
        ACCEPT : */*
    entry_id : 1022
    time : 2017-07-05-11:59:32
    url  : /api/philips2/hue-
           link/473c83888ad23556f4633893c40325f7
    remote_ip : 172.16.10.2
    type : POST
  ▼ reply_content {1}
        detail : Method \"POST\" not allowed.
```

Figure 4.11: Example of one entry of the REST honeypot log file, in Json format parsed by *Json Online* [3]

4.5.2. The XMPP Log System

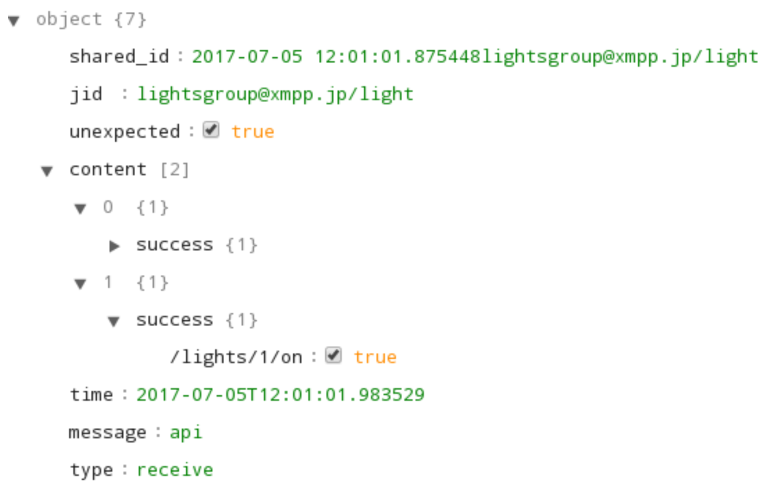
XMPP Log System of the honeypot also has two classes of logs.

Since the XMPP honeypot is based on *sleekxmpp* and *phue-lib*, one class of the log is the logs generated by them, showing the status and information about this process. It is called XMPP system log. The log system saves one log file for each JID, named by JID, which is unique and will not be overwritten. A directory named by *"syslog + TIMESTAMP.log"* under the *filepath* directory given in the configuration file will be made each time at the start of running this honeypot. The system log files are saved in this directory.

Another kind of log is similar to the requests log of the REST honeypot. It is in Json format and saved in a file that each line is a Json entry. It is called XMPP traffic log. Because it logs down all the traffics including the incoming and outcoming messages through XMPP chat or XML stream to the server and the interaction to the API (HTTP request and response). Figure 4.12 shows one example of one Json entry from the XMPP traffic log. The Json entry includes:

- *shared_id*: This is used for cooperation with the REST log in further data analysis. Details are explained in next section.
- *jid*: This indicates from which JID account is the log generated. Since all the traffic are logged in the same file whatever the JID is, this item is used for distinguishing.
- *unexpected*: Boolean. It is generated by the first analysis by the XMPP-client honeypot. A *True* means that this traffic is potentially an attack from the attacker, which needs for further analysis.
- *content*: This is the content of the traffic. If the traffic is an XMPP message, then this would be the content of the message. If the traffic is the HTTP response. Then it is the responded data from the API.
- *time*: The time when the log is generated.

- *message*: Either "xmpp" or "api", indicating where is this traffic from or to.
- *type*: Either "receive" or "send", indicating if the traffic is from other objects or sent by XMPP client honeypot.
- *(mode)*: This only exists when it is an HTTP request sent by XMPP client honeypot. *mode* is the type of HTTP request like "PUT", "POST", "GET", etc.
- *(address)*: This only exists when it is an HTTP request sent by XMPP client honeypot. *address* is the URL of the HTTP request.



```
▼ object {7}
    shared_id : 2017-07-05 12:01:01.875448lightsgroup@xmpp.jp/light
    jid : lightsgroup@xmpp.jp/light
    unexpected : ✓ true
  ▼ content [2]
    ▼ 0 {1}
      ▶ success {1}
    ▼ 1 {1}
      ▼ success {1}
          /lights/1/on : ✓ true
    time : 2017-07-05T12:01:01.983529
    message : api
    type : receive
```

Figure 4.12: Example of one entry of the XMPP traffic log file, in Json format parsed by *Json Online* [3]

4.5.3. The "shared_id" Method

A "shared_id" is used for cooperation of the REST and XMPP honeypot in the log system. As mentioned before, the JID on the XMPP client honeypot is linked to the "smart bulb" on the REST honeypot. So we can see that the log system of REST and XMPP should not be totally independent. A method o "shared_id" is proposed and applied here to link the log entries between the REST Json log and the XMPP Json log.

While receiving a message or a traffic from the XMPP part, the log system is going to generated a "shared_id" that is based on the related JID and the current timestamp. The format of the "shared_id" is *TIMESTAMP+JID*. An example is shown on Figure 4.12. Then this generated "shared_id" is written in the header of the HTTP request to the API and transferred to the REST honeypot. Then when the REST honeypot received this request, header is parsed and the "shared_id" is saved in the Json entry of the REST Json log. Thus the logs are correlated by the "shared_id".

5 Data Analysis and Results

In this chapter, the collected data is analyzed to find some patterns and rule about the attacker's activities targeting the IoT platform. This could also prove that the ThingPot has the ability to contribute to the IoT Platform security.

5.1. Introduction

The data analysis is mainly based on the logs of each node. The honeypot is running from 22th June to 7th August. During this time, logs from each node are taken. However, by observing the logs on XMPP node, no relevant data were found (further discussion is addressed on next chapter). Thus in the data analysis, the REST logs from the three API node 1 2 3 will be taken into consideration as the main source. These data has two kinds: the proxy log and the API log. The proxy is installed on Node 1 and 2. So the proxy log could not reflect the behaviour the node 3. While all the three nodes have the API log. However, the API log does not cover all the information in proxy since it could only handle the HTTP request. Moreover, the API log is not built up well at the very beginning, but the proxy log is working all the time. Thus both these two logs will be used to work together for complementary and correspondence.

5.2. Data Analysis Methodology

The data analysis methodology is shown in Figure 5.1. There are three stages in the data analysis:

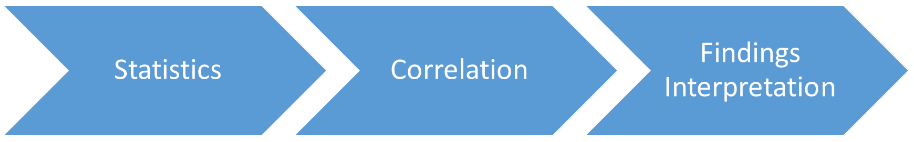


Figure 5.1: The Data Analysis Methodology

- Statistics: Firstly, tables and pictures of the statistics are made from the raw data (logs) to have an initial understanding on what happened in the honeypot. It involves the classification, countings and simple interpretation of data. This stage helps to give clues and understandings. The statistics are the dataset for the stage of correlation.

- Correlation: Secondly, correlations between the statistics presented on the first stage are made to find more information about the attackers and attacks. Instead of analyzing each part of the statistics separately, the correlation is to find the relationship among all the information that is given. This would be pretty interesting because it could tell more stories.

- Findings Interpretation: Thirdly, after doing the correlation, the understandings of the honeypot data would be clearer. Thus a summary could be made and there would be several findings to interpret. Attacks could be listed based on the statistics and correlations.

5.3. Statistics

This section is going to present all the relevant statistics that would contribute to further correlations and interpretations. Classifications in general are done in the first two sections. Then in the section 5.3.3, a number of tables and a simple interpretation for each table are given. These tables show the top number of each kind of items that are useful for the correlation and interpretation part. These kind of tables are call TOP tables in this thesis.

5.3.1. Targeted requests

By having a look at all the requests, it could be classified by three kinds: targeted, undefined, not-targeted. "Targeted" request means that it could be seen that the request for something specific instead of a general scanning. These targeted requests might be based on some knowledge about this implementation in advance. "Undefined" requests means it is not clear whether it is targeted or not, further analysis is needed. From the proxy log, the total number of requests is 61,637. From the log of node 3, the total number of requests is 52,104. Thus the total number of all the requests on the honeypot is 113,741. Table 5.1 shows the numbers of targeted, undefined and non-targeted requests and the percentages. In the column of "Targeted?", a ✓means that it is a targeted request. A ✗means the request is a normal scanning. A "-" means it is not clear if it is targeted or not. From this we can see that around half of the requests can be seen as targeted.

Targeted?	Count	Percentage
✓	47,297	41.5%
-	10,444	9.2%
✗	56,000	49.2%
TOTAL	113,741	

Table 5.1: Distribution of requests classified by targeted or not

The other half are just common scanning to the services. The targeted requests are more interesting because this means the attacker might be looking for a smart device or some known vulnerabilities.

5.3.2. The HTTP REST requests that start with "/api"

Since the more interesting data is mainly HTTP REST requests, then the analysis on the HTTP REST requests gives more insight about attacker's activities. The requests starts with "/api" are more likely to be a targeted attack than scanning. Because all the valid URLs that are defined by the API honeypot start with "/api". And this is based on the knowledge of the Phue bridge.

"/api"?	Count	Percentage
✓	48705	42.9%
✗	64760	57.1%
TOTAL	113465	

Table 5.2: Distribution of requests classified by targeted or not

5.3.3. TOP tables

In this section, statistics of requests, status code (for HTTP requests), user agent, referrer, source IP address and request type will be counted and classified. Tables of the top for each subject are made and shown with simple interpretations.

Requests

Table 5.3 shows the top 10 requests by taking all the logs into consideration. Some of the entries that have similar patterns are merged together to record the number in total. The "X" in the column of "Request" could be any character.

Count	Request
31568	"POST /api/ HTTP/1.1"
5175	"GET /sfi9876.XXXXXXX HTTP/1.1"
3014	/http:/84.19.176.29:80/PMA201X/
2984	"POST /api/list/ HTTP/1.1"
1659	"POST / HTTP/1.1"
891	"GET / HTTP/1.1"
622	"GET http://testp3.pospr.waw.pl/testproxy.php HTTP/1.1"
520	"GET /api.XXXX/ HTTP/1.1"
342	"GET / HTTP/1.0"
280	/http:/84.19.176.29:80/mysql/admin/

Table 5.3: Top 10 requests from all nodes' logs

A list of URLs that start with "/api" is made. Table 5.4 shows the tops 10 URLs that start with "/api", which is highly possible that it is targeted. In this table, the "-" under the column "Remark" means it is not sure if every request is targeted. But of course it is highly possible because it starts with "/api". The "targeted" under item "Remark" means it is for sure a targeted attack. In this table we can see that the keyword "hue", "philips" is included. "belkin" and "wemo" are also very important keyword because Wemo from Belkin is another popular smart home devices brand. These URLs show that Smart Devices is the target of the attackers.

Count	URL	Remark
31607	/api/	-
2984	/api/list/	-
25	/api/config	targeted
7	/api/philips1/hue/8a92a9360fac1de8ada3903dce0795ed	targeted
6	/api/belkin/wemo/af3bdcebd800931357951db376e0dad7	targeted
5	/api/belkin/wemo/af3bdcebd800931357951db376e0dad7/	targeted
4	/api/‘false‘	-
4	/api/‘uname‘	-
4	/api/‘true‘	-
4	/api/‘echo%20skip12“echo%2034fish‘	

Table 5.4: Top 10 HTTP URLs started with "/api" taken from the proxy logs

In the Appendix, more tables in different ways of classification are shown. By only analyzing the HTTP requests, Table A.1 shows the top 11 HTTP URLs among all of the requests. By ignoring the HTTP requests, Table A.2 shows the top ten non-HTTP requests logged by the proxy. From these we can see the frequent non-HTTP requests. From the count number we can see that the number is not very large, which means the requests might be made manually. Decoding is needed for further details.

Source IP addresses

IP address is a very important parameter to analyze. The IP address itself can already contribute to the IoT security. Moreover, many information could be get from IP address that the attacker is using. Table 5.5 lists the top 20 IP that is used and logged by the proxy, including all the services.

As interesting finding it was found that TOR network (platform for anonymous communication [11]) is used by some attackers in order to hide the real source IP address. The IP addresses in Table 5.5 are checked to see if they are in the TOR relay list. The column "TOR?" shows if the IP address is using TOR technology. “Yes” and "No" means it is using TOR or not respectively.

Count	IP	TOR?
4982	193.70.95.180	Yes
3810	93.115.95.207	Yes
1669	104.223.123.98	Yes
1191	163.172.101.137	Yes
1060	192.42.116.16	Yes
889	89.234.157.254	Yes
861	79.137.79.167	Yes
834	91.219.237.244	Yes
736	79.172.193.32	Yes
717	204.85.191.30	Yes
694	94.242.246.24	Yes
671	94.242.246.23	Yes
669	91.223.82.156	Yes
663	5.254.79.66	Yes
652	78.109.23.1	Yes
622	91.196.50.33	No
610	176.126.252.11	Yes
572	216.218.222.13	Yes
565	163.172.212.115	Yes
564	185.170.42.4	Yes

Table 5.5: Top 20 source IP addresses from all requests taken from the proxy logs

Moreover, it is also interesting to see what are the top IPs that are used in HTTP requests respectively. Table 5.6 is the top 10 IP addresses that is used in HTTP request.

Count	IP
4962	193.70.95.180
3803	93.115.95.207
1668	104.223.123.98
1191	163.172.101.137
1060	192.42.116.16
889	89.234.157.254
861	79.137.79.167
834	91.219.237.244
736	79.172.193.32
717	204.85.191.30

Table 5.6: Top 10 source IP addresses from HTTP requests taken from the proxy logs

In the Appendix, Table A.3 is a list of top 20 IP that is based on the API log to make it complete. The API log only logs the HTTP requests, but it contains logs of node 3. These two tables together could compensate each other and a list of IP addresses that are used by the attacker could be made. Table A.4 is the top 10 IP addresses that is used in non-HTTP request.

From the list of IP addresses, more information could be get. Table 5.7 shows the list of top 5 countries that the IPs are from. It shows that the top five countries are: US, China, France, Netherlands and Russia.

In the Appendix, more tables are shown. Autonomous system is a number of IP prefixes which have the same and well defined routing policy, coming from one or more network operators. [33]. Autonomous System

Count	Country
143	US
67	CN
40	FR
36	NL
25	RU

Table 5.7: Top 20 countries that IP addresses is located

Number (ASN) is a number officially registered by the Internet service provider (ISP). Table A.5 shows the top 20 ASNs of the IPs. Table A.6 shows the Regional Internet Registry (RIR) of the IP addresses. A Regional Internet Registry is an organization that in charges of the Internet Number's allocation and registration. The RIR is divided by the physical area of the world. The Internet Number includes the IP and the ASN.

5.3.4. HTTP Statics

Since most of the requests are HTTP requests. It is important and helpful to do analysis of the HTTP requests. The important parameters are: the type, status code, user agent, referrer:

- Type:
  Table 5.8 is the number of HTTP request counted by type.

Count	HTTP Type
50776	HEAD
38749	POST
23708	GET
307	PUT
9	CONNECTED
4	PROPFIND
4	OPTIONS

Table 5.8: Counts of each type of HTTP requests taken from the proxy logs

Count	Status Code	Remark
34702	200	OK. Request succeed.
5960	499	Token Required. This is not an official codes. It means that a token is required but was not submitted
65666	404	Not Found. The requested resource (URL) could not be found.
2477	502	Bad Gateway. The proxy received an invalid request.
2419	405	Method Not Allowed. This means the URL is valid, but with a wrong type.
2224	301	Moved Permanently. The request should be redirected to a new URI.
320	400	Bad Request. The server cannot or will not process the request due to an apparent client error (e.g., malformed request syntax, size too large, invalid request message framing, or deceptive request routing). This mean the requests might contains some weird things that the honeypot can not handle.
85	304	Not Modified. This is a redirection from the proxy that the client (API) will take charge of response.
41	504	Gateway Timeout. The proxy did not receive a timely response from the upstream server.
4	302	Found. For URL redirection.
1	500	Internal Server Error.

Table 5.9: Number of HTTP status code of responds taken from the proxy logs

- Status Code:
  The status code of HTTP is the code that represents different status of this request according to the API

configuration. Each of the code has its own definition. Table 5.9 lists the rank of the HTTP status code of the incoming requests.

- User Agent:
  User agent is a very important information for the API honeypot data analysis. It is a field of the HTTP request that contains the information of application type, operating system, software vendor or software version of the requesting software user agent. Thus from this field, more information of the requester could be get to analyze. It could be seen as a relatively unique id to represent a pattern in these logs and during this analysis process. Table 5.10 lists the top 20 user agents of all of the incoming requests. Under the column "Remark", a quick search is done to have a first impression. In the Appendix,

Count	User Agent	Remark
48460	Mozilla/5.0 Jorgee	A malware that scans vulnerabilities on web
31567	"shooter"	No information is found
9229	"Mozilla/5.0 SF/2.10b"	Default user-agent for Skipfish version 2.10b [18]
2984	"botlight"	???
2378	"000modscan"	modscan: A Scada Modbus Network Scanner
1867	"httpget"	no information is found
831	"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"	from web browser IE 11 on a Windows 10 OS
622	"Mozilla/5.0 (Windows NT 5.1; rv:32.0) Gecko/20100101 Firefox/31.0"	From a web browser firefox on a Windows XP OS
607	"0000modscan"	modscan: A Scada Modbus Network Scanner
313	"Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko"	A web browser IE 11 on a Windows 7 OS
275	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:47.0) Gecko/20100101 Firefox/47.0"	a web browser firefox on Mac OS X 10.11
248	ioscan	No information was found
236	"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.2) Gecko/2008092809 Gentoo Firefox/3.0.2"	
186	"Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko"	
158	"Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko"	
154	"Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:map_codes)"	Nikto: A web scanner tool
148	"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome /59.0.3071.115 Safari/537.36"	
110	"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome /58.0.3029.110 Safari/537.36"	
103	"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; Touch; LCJB; rv:11.0) like Gecko"	
96	Python-urllib/2.7	The library urllib is used.

Table 5.10: Top 20 User Agents from all requests taken from the proxy logs

to make classification of HTTP and non-HTTP requests, Table A.7 shows the top 10 user agents of HTTP requests. And Table A.8 shows the top 10 user agents of non-HTTP requests. To make further classification, the HTTP requests can be divided into two kinds: URLs start with "/api" and the others. Table A.9 is the top 10 user agents from the requests whose URLs start with "/api". Table A.10 is the top 10 user agents from the rest HTTP requests.

- Referrer: Referrer is a field in HTTP request. It identifies the web page address that linked to the requested resource. It reflects that the attacker might reach the resource through a link on a web page. It contains useful information that can contribute to the data analysis of the honeypot. Table A.11 in the Appendix is a list of referers of the incoming requests taken from the API logs. From this list we can see that some of the related URLs are set as a link on any web page of the attackers.

5.4. Correlation

In this section, further analysis is done according to the given tables in section 5.3. Correlations between the statistics that is useful will be find out.

5.4.1. Requests/URLs Analysis and Correlations with Other Statistics

There are a lot of things we can get from the URL list.

By observing Table 5.4, we can see that the targeted requests more likely happen in the URL starts with "/api". And there are URLs that involved keywords like "Philips", "hue" in that table. By checking the details of the logs, it is found that there are a large number of URLs that have very similar prefix that contains interesting keyword. While only the last part (which is some fixed-length random string) is different. Thus a further statistics analysis is done and the result is shown in Table 5.11. In this table, the column "URL" is not a specific URL but a pattern. It counts the number of URLs whose prefix part is the same. It could be seen that the top URLs in this table is obviously targeting some resources targeting to the Philips Hue devices or Belkin Wemo devices. And the string after the prefix is always a 32-digit string, which is likely to be a MD5 encryption. This at least proves that the attackers are looking for smart devices. **Correlation with user agent and Status Code**

Count	URL
1258	/api/philips/hue
522	/api/hue
396	/api/philips2/hue-link
382	/api/belkin/wemo
306	/api/philips1/hue
4	/api/index
4	/api/resource

Table 5.11: URLs start with "/api" in targeted pattern taken from the API logs

It is interesting to see the correlations between the URLs and user agent. To have a look at some of the URLs list in Table 5.4 and Table 5.11, something interesting can be found. Table 5.12 shows the user agent of some of the interesting URLs. It is not complete but we can see some rules there that gives clues.

1. Most of the "/api/" requests are from "shooter". By checking the number of requests from "shooter" in Table 5.10, we can see that all the "shooter" requests are "/api/". And almost all are successful with a 200, this is because of the settings of the honeypot.

2. All the requests of "/sfi9876.XXXXXXX" is from the user agent "Mozilla/5.0 SF/2.10b". This user agent is the *Skipfish*. Here the conclusion of this request can be made that the attacker is using *Skipfish* as the tool to scan. Most of the request is a 404, which means the general scannings are not easy to hit the correct URL.

3. All of the "/api/list/" requests are from "botlight". By checking the number of requests from "shooter" in Table 5.10, we can see that all the "botlight" requests are "/api/list". This URL is not valid in the API, thus the status code are all 404.

4. It is a reasonable guess that "000modscan" and "0000modscan" and even "curl/7.52.1" are from the same person or follow the same pattern and they are giving similar requests to the honeypot. Taking a look at all the status code it has, most are 502, 200 and 405. A 502 means the API might be down at that time. 405 means that the URL is valid, only the method is not allowed. This makes sense that this is not a generic scanning. Thus it could hit the valid URL but wrong method. We can also see a lot of 200,

Count	URL	User Agent	Status Code
31607	/api/	"shooter" : 31567 "Mozilla/5.0" : 35 "wget/1.16" : 2	200:31604 502: 3
5175	/sfi9873.XXXXXXXX	"Mozilla/5.0 SF/2.10b": 5175	404: 5384 200:2
2984	/api/list/	"botlight": 2984	404: 2984
1258	/api/philips/hue/{32_chars}	"000modscan": 1158 "0000modscan": 100	502: 941 405: 216 200:100 400: 1
499	/api/hue/{0-750}	"null": 251 "ioscan": 248	
440	/api/phi/light/{32_chars}	"000modscan": 304 "0000modscan": 100 "httpget": 36	
396	/api/philips2/hue-link/{32_chars}	"000modscan": 234 "0000modscan": 100 "curl/7.52.1": 81	405:296 200: 100 502:18 400:1
382	/api/belkin/wemo/{32_chars}	"000modscan": 200 "0000modscan": 106	301: 386 200: 5 400: 1
306	/api/philips1/hue/{32_chars}	"000modscan": 200 "0000modscan": 106	405: 200 200: 106
300	/api/tplink/light/{32_chars}	"000modscan": 200 "0000modscan": 100	301: 300
30	/api/device//{32_chars}	"mass": 30	

Table 5.12: Some interesting URLs and it's user agent and status code

which means a successful request. The number of 200 in these requests is much larger than the 200 in general scanning. From this we can see that a targeted scanning has a higher possibility to reach the resource.

5.4.2. IP addresses

Among all the incoming requests in the three nodes, there are around 1383 different IP addresses is involved. IP address is a pretty important parameter to analyze because it could give information like the location, the ASN, etc. While doing the analysis of the IP addresses. It if found that a lot of IP addresses are using TOR network. It is important to know if the TOR technology is used or not. Because if the TOR technology is used, then it is not the actual source. But if the TOR is not used, then the IP address could be seen as the actual resource IP. Among the IPs shown in Table 5.5, most IP are using TOR technology. *91.196.50.33* and *196.54.55.13* is not on the TOR relay list, which means they can reflect the real source of the attacker. *91.196.50.33* is from Radomsko, Lodzkie of Poland, whose ISP is Euronet S.C. Jacek Majak Aleksandra Kuc. It requested *GET http://testp3.pospr.waw.pl/testproxy.php HTTP/1.1* to the honeypot and has been reported for the same reason before. *196.54.55.13* is from Cedar Falls, Iowa of United States whose ISP is LogicWeb Inc. It generated a lot of POST requests with very specific payload, with the user agent "shooter". It will be introduced later. This IP address was reported as spam and brute force attacks before.

Correlation with Table 5.7

Actually the Table 5.7 is taken based on Table 5.5 to see the countries distribution. But if the TOR network is used, which in this case the answer is yes, then the Table of countries is not reliable. The same as Table A.5 and Table A.6.

5.4.3. Request Type

From Table 5.8 we can see that the POST and GET are very common types that the attacker will use. By using POST, new items could be added. And GET could get the data from the API. This means the attackers are interested in modifying, adding or getting values of the target API.

Correlation with Status Code

It is interesting and valuable to analysis what is the status code of each type of HTTP request. Table 5.13 shows the status code of each type of the HTTP requests. From this we can see what are the status of each kind of requests. Most of the HEAD requests have a 404, which means the URL is not valid. But still there are some 200. However, most of the POST requests succeed with a 200.

Count	HTTP Type	Status Code
50776	HEAD	404:50753
		200: 18
		302: 4
		400: 1
38749	POST	200:31572
		502: 956
		404:2992
		405: 2401
		301: 782
		400: 5
23708	GET	404:10526
		499: 5960
		200: 2785
		502: 1512
		301: 1434
		304: 49
		400: 55
		504: 41
		405: 5
		500: 1
307	PUT	200: 278
		404: 29
9	CONNECTED	?
4	PROPFIND	404: 4
4	OPTIONS	404: 4

Table 5.13: Counts of each type of HTTP requests taken from the proxy logs

5.4.4. User Agent

The user agent can somehow be seen as the unique id for the data that are caught. Since each user agent might perform differently. But the requests that shares the same user agents more likely to have similar performance. Correlating the user agent with other values could be helpful to understand the attacks.

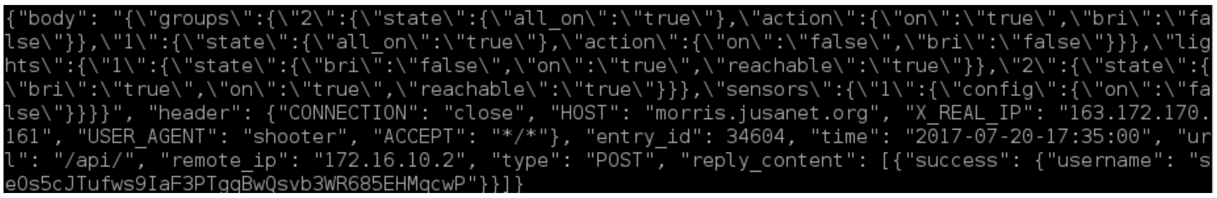
By checking the Table 5.10, we can see that a lot of requests come from user agents who look like from a web browser. This means that these requests might come from a web browser manually by the attacker. By checking into details about the frequency and the amount. It could be a guess that the requests might come from some plugins of a web browser. Or the attacker is using this way to hide the real user agent.

Correlation with Request Type and Status Code

Table 5.14 shows the correlation between the user agent and the request type and number of IPs. Each line is one user agent, with the number of types of this user agent and how many IPs are used by this user agent. The URL or URL pattern of the user agent is also very interesting. By combing the result of Table 5.12 and Table 5.14 as well as checking details (URLs, body) of part of the log, a summary based on each user agent can be given:

- Mozilla/5.0 Jorgee:
  "Jorgee" is a malware that tries to make use of the sql web admin flaws [68]. It appears 48460 times in total with 360 IPs involved, lasting from 28th June to 2nd August and happens everyday. Each IP only sent around 200 to 400 requests. All the request types are HEAD. The URLs have keyword "db", "admin", "pma", "php", "sql", "web", "database" and "my". The URL is the permutation and combination of these word. This user agent only appears on node 3.

- shooter:
  There is no much information about "shooter" on the Internet. This means this user agent is created manually. It generated 31567 requests on the honeypot, with 92 IPs involved. TOR technology is used to hide the real source IP. The requests start from 19th July 23:31 to 20th July 17:35, coming continuously. The requests come every 1 to 4 seconds. All the requests' URLs are "/api/" with the POST method and a specific body content. Figure 5.2 shows a sample of the log entries which contains all the relevant information of this request. From this picture we can see that the body content is very interesting. The body content is in a well organized format that is following the format of the Philips Hue's data structure. A replay of this request is done on the real devices (Philips Hue White). The reply from the real Phue bridge is an error message saying that the parameter is no available.

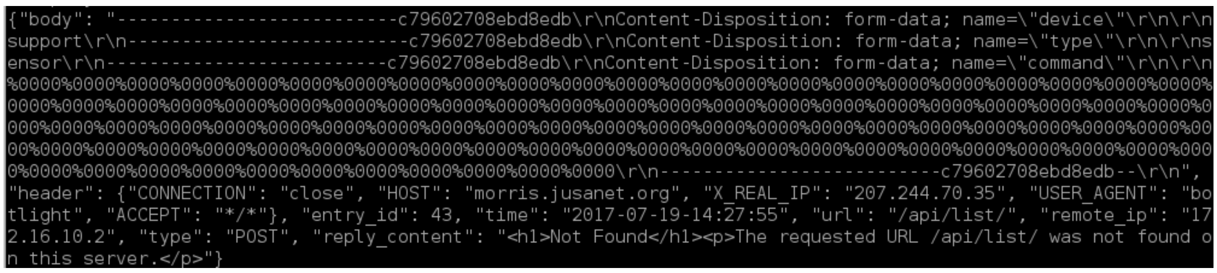


```
{"body": "{\"groups\":{\"2\":{\"state\":{\"all_on\":\"true\"},\"action\":{\"on\":\"true\",\"bri\":\"fa
lse\"}},\"1\":{\"state\":{\"all_on\":\"true\"},\"action\":{\"on\":\"false\",\"bri\":\"false\"}}},\"lig
hts\":{\"1\":{\"state\":{\"bri\":\"false\",\"on\":\"true\",\"reachable\":\"true\"}},\"2\":{\"state\":{
\"bri\":\"true\",\"on\":\"true\",\"reachable\":\"true\"}}},\"sensors\":{\"1\":{\"config\":{\"on\":\"fa
lse\"}}}}", "header": {"CONNECTION": "close", "HOST": "morris.jusanet.org", "X_REAL_IP": "163.172.170.
161", "USER_AGENT": "shooter", "ACCEPT": "*/*"}, "entry_id": 34604, "time": "2017-07-20-17:35:00", "ur
l": "/api/", "remote_ip": "172.16.10.2", "type": "POST", "reply_content": [{"success": {"username": "s
e0s5cJTufws9IaF3PTgqBwQsvb3WR685EHMqcwP"}}]}
```

Figure 5.2: One of the log entries whose user agent is shooter

- Mozilla/5.0 SF/2.10b:
  This is the default user-agent for Skipfish [18]. 7 IPs is involved with 9229 requests in total. The requests are all 20115 GET, 47 PUT and 47 FOO. Skipfish is a scanning tool for security of web application. It was designed for security checks. But it can also be made use of by the attackers.

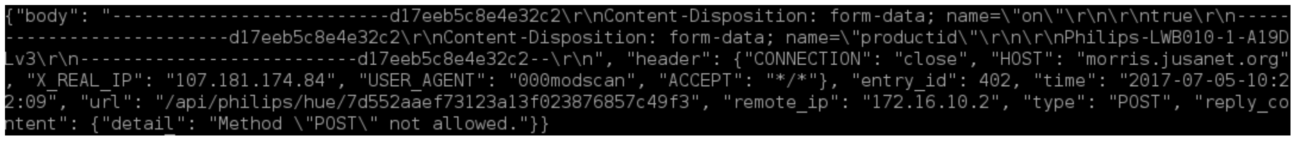
- botlight:
  2984 requests are from this user agent. The burst of requests started from 19th July 14:27 till 19th July 15:50. All the requests are POST requests with the same URL "/api/list/". 21 IPs is involved and TOR technology is used to hide the real source IP. Figure 5.3 shows one of the log entries whose user agent is "botlight". It is interesting that the body content of the requests is following the *multipart/form-data*, which is used for file upload. But there are also a lot of % and 0s, which looks like a fuzzing.



```
{"body": "--------------------------c79602708ebd8edb\r\nContent-Disposition: form-data; name=\"device\"\r\n\r\n
support\r\n--------------------------c79602708ebd8edb\r\nContent-Disposition: form-data; name=\"type\"\r\n\r\ns
ensor\r\n--------------------------c79602708ebd8edb\r\nContent-Disposition: form-data; name=\"command\"\r\n\r\n
%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%
0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0
000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%00
00%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%000
0%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000%0000\r\n--------------------------c79602708ebd8edb--\r\n",
"header": {"CONNECTION": "close", "HOST": "morris.jusanet.org", "X_REAL_IP": "207.244.70.35", "USER_AGENT": "bo
tlight", "ACCEPT": "*/*"}, "entry_id": 43, "time": "2017-07-19-14:27:55", "url": "/api/list/", "remote_ip": "17
2.16.10.2", "type": "POST", "reply_content": "<h1>Not Found</h1><p>The requested URL /api/list/ was not found o
n this server.</p>"}
```

Figure 5.3: One of the log entries whose user agent is botlight

- 000modscan:
  000modscan only have POST request with 12 IPs involved. The requests happened from 5th July 10:22 to 11:59 The URL follows the pattern that has been shown in Table 5.12, which are */api/philips/hue/{32_chars}*, */api/philips2/hue-link/{32_chars}*, */api/belkin/wemo/{32_chars}* and */api/philips1/hue/{32_chars}*. These POST requests have very interesting body content, which is similar to the body of "botlight". One sample request is shown in Figure 5.4. From this picture we can see that this body content is also following the *multipart/form-data*. Moreover, in the body content, some of the word is related to the Philips Hue device.



```
{"body": "--------------------------d17eeb5c8e4e32c2\r\nContent-Disposition: form-data; name=\"on\"\r\n\r\ntrue\r\n-----
---------------------d17eeb5c8e4e32c2\r\nContent-Disposition: form-data; name=\"productid\"\r\n\r\nPhilips-LWB010-1-A19D
Lv3\r\n--------------------------d17eeb5c8e4e32c2--\r\n", "header": {"CONNECTION": "close", "HOST": "morris.jusanet.org"
, "X_REAL_IP": "107.181.174.84", "USER_AGENT": "000modscan", "ACCEPT": "*/*"}, "entry_id": 402, "time": "2017-07-05-10:2
2:09", "url": "/api/philips/hue/7d552aaef73123a13f023876857c49f3", "remote_ip": "172.16.10.2", "type": "POST", "reply_co
ntent": {"detail": "Method \"POST\" not allowed."}}
```

Figure 5.4: One of the log entries whose user agent is 000modscan

- httpget:
  The requests whose user agent is httpget are all HTTP GET with 7 IPs involved. The requests happened in two period: one on 5th July from 12:02 to 12:18, one on 6th July from 11:34 to 11:46. The URLs are following three patterns: */api/phi/light/{32_chars}/tokens*, */api/{32_chars}/tokens* and */api/{32_chars}*. *{32_chars}* means a random 32 long string which contains only digits and low-case letters. Since both two days has the URL whose pattern is */api/32_chars/tokens*, it could be assumed that all the requests with "httpget" are from one source. This could be seen as a targeted attack.

- "Mozilla/5.0(WindowsNT5.1;rv:32.0)Gecko/20100101 Firefox/31.0":
  From the table we can see that all the requests are HTTP GET with 14 IPs involved. While most of the requests are from IP 91.196.50.33. The requests are from 28th June to 31st July, 2 to 5 times per day. Most of the requests' URL is */http:/testp3.pospr.waw.pl/testproxy.php*. This is a URL that is looking for open proxies.

- 0000modscan:
  All the requests whose user agent is "0000modscan" are GET requests with 12 IPs involved. The requests happened from 5th July 09:37 to 09:42. The URL of the requests are */api/tplink/light/{32_chars}*, */api/philips/hue/{32_chars}*, */api/phi/light/{32_chars}*, */api/philips2/hue-link/{32_chars}*, */api/belkin/wemo/{32_chars}*, */api/philips1/hue/{32_chars}*. This is a targeted HTTP GET scan targeting to the smart light devices.

- "Mozilla/5.0 (Macintosh;Intel Mac OS X10.11; rv:47.0) Gecko/20100101 Firefox/47.0":
  The requests are only from two IPs: 183.129.160.229 (193 times) and 60.191.38.77 (82 times). All are GET with the URL /. This user agent means that it is from the Mac Operating system with a Firefox web browser. By checking the time of the requests, it is likely to be sent manually. Thus a guess is that, this is somebody who found the proxy and requested the data resource manually.

- ioscan:
  All the requests whose user agent is "ioscan" are HTTP GET with 2 IPs involved. The requests happened on 4th July from 13:58 to 14:00. All the URL follow the pattern */api/hue/{0-216}*, where {0-216} is a number ranges from 0 to 216. This can be seen as a targeted scan since the "hue" is a keyword of the honeypot.

- Python-urllib/2.7:
  Python-urllib/2.7 is a Python library for fetching data across the World Wide Web [9]. The requests whose user agent is "Python-urllib/2.7" are all HTTP GET with 3 IPs involved. 94 of the requests happened from 1st July 01:10:09 to 01:10:19, which means the 96 requests happened in 10 seconds. And they are all from one IP (185.77.172.42). From this we can see that there must be a scripts running instead of manually from one attacker who is trying to scan the honeypot. By checking the URL of these requests, it could be found that the URL is quite random but in general is looking for vulnerabilities of the sql web admin.

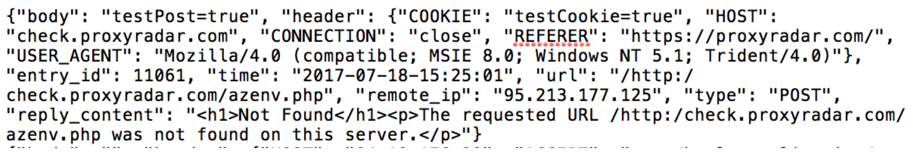
Count	User Agent	Type of Request	Number of IP
48460	Mozilla/5.0 Jorgee	HEAD: 48460	360
31567	"shooter"	POST: 31567	92
9229	"Mozilla/5.0 SF/2.10b"	GET: 9171 PUT: 29 FOO: 29	7
2984	"botlight"	POST: 2984	20
2378	"000modscan"	POST: 2378	12
1867	"httpget"	GET: 1867	7
622	"Mozilla/5.0 (Windows NT 5.1; rv:32.0) Gecko/20100101 Firefox/31.0"	GET: 622	14
607	"0000modscan"	GET: 607	12
275	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:47.0) Gecko/20100101 Firefox/47.0"	GET: 275	2
248	ioscan	GET: 248	2
96	Python-urllib/2.7	GET: 96	3

Table 5.14: Top 20 User Agents from all requests taken from the proxy logs

5.4.5. Interesting Body Content

The inspiration of this section is taken while doing analysis and correlation of the user agent. Interesting things might be found in the payload (e.g. random strings for fuzzing, specific payload for specific purpose related to the device, etc.)

1. "testPost=true":
   Figure 5.5 shows one sample of the log entries of these requests. In those requests whose body is "testPost=true", the URL is */http:/check.proxyradar.com/azenv.php* and the referer is *https://proxyradar.com/*. This is a tool for searching open proxies.



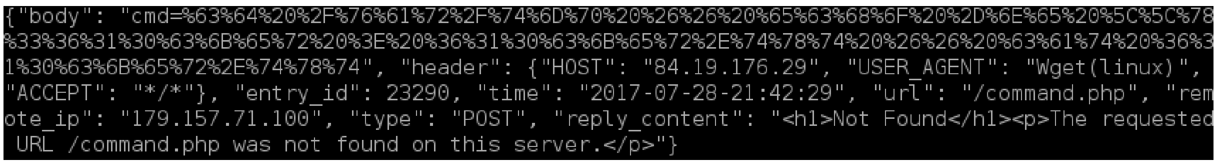
```
{"body": "testPost=true", "header": {"COOKIE": "testCookie=true", "HOST":
"check.proxyradar.com", "CONNECTION": "close", "REFERER": "https://proxyradar.com/",
"USER_AGENT": "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)"},
"entry_id": 11061, "time": "2017-07-18-15:25:01", "url": "/http:/
check.proxyradar.com/azenv.php", "remote_ip": "95.213.177.125", "type": "POST",
"reply_content": "<h1>Not Found</h1><p>The requested URL /http:/check.proxyradar.com/
azenv.php was not found on this server.</p>"}
```

Figure 5.5: One of the log entries which contains the body content "testPost=true"

2. This is an encoded request (shown in Figure 5.6) whose body content after translation is:

$$cd/var/tmpecho-ne\\x3610cker > 610cker.txtcat610cker.txt$$

   According to [34], this is looking for vulnerable routers on the Internet.



```
{"body": "cmd=%63%64%20%2F%76%61%72%2F%74%6D%70%20%26%26%20%65%63%68%6F%20%2D%6E%65%20%5C%5C%78
%33%36%31%30%63%6B%65%72%20%3E%20%36%31%30%63%6B%65%72%2E%74%78%74%20%26%26%20%63%61%74%20%36%3
1%30%63%6B%65%72%2E%74%78%74", "header": {"HOST": "84.19.176.29", "USER_AGENT": "Wget(linux)",
"ACCEPT": "*/*"}, "entry_id": 23290, "time": "2017-07-28-21:42:29", "url": "/command.php", "rem
ote_ip": "179.157.71.100", "type": "POST", "reply_content": "<h1>Not Found</h1><p>The requested
 URL /command.php was not found on this server.</p>"}
```

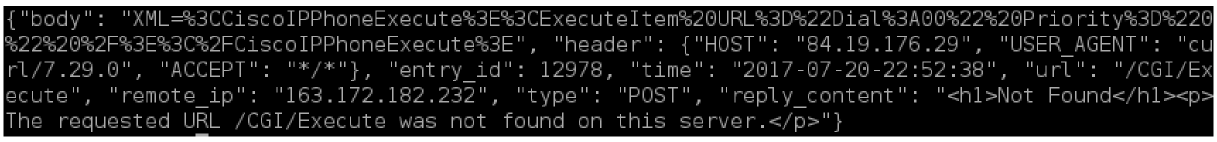
Figure 5.6: One of the log entries which contains the body content "cmd=%63%64..."

3. This is an encoded request (shown in Figure 5.7) whose body content after translation is:

$$XML = \backslash < CiscoIPPhoneExecute\backslash > \backslash < ExecuteItem\backslash URL\backslash = \backslash"Dial$$

$$\backslash :00\backslash"\backslash Priority\backslash = \backslash"0\backslash"\backslash\backslash/\backslash > \backslash < \backslash/CiscoIPPhoneExecute\backslash >$$

   It seems like a malware that is attempting to get access to phpmyadmin/scripts/setup.php [41].



```
{"body": "XML=%3CCiscoIPPhoneExecute%3E%3CExecuteItem%20URL%3D%22Dial%3A00%22%20Priority%3D%220
%22%20%2F%3E%3C%2FCiscoIPPhoneExecute%3E", "header": {"HOST": "84.19.176.29", "USER_AGENT": "cu
rl/7.29.0", "ACCEPT": "*/*"}, "entry_id": 12978, "time": "2017-07-20-22:52:38", "url": "/CGI/Ex
ecute", "remote_ip": "163.172.182.232", "type": "POST", "reply_content": "<h1>Not Found</h1><p>
The requested URL /CGI/Execute was not found on this server.</p>"}
```

Figure 5.7: One of the log entries which contains the body content "XML=%3CCiscoIPPhoneExecute..."

5.5. Findings Interpretation

After gathering information from the logs, a summary could be made to summarize the information shown in the last two sections. Attacks could be list according to the TOP tables and correlations. However, it could not be guaranteed as a complete list of attacks found on this honeypot since the data base is large and the time to do the analysis is limited.

5.5.1. Targeted Attack that is Trying to Take Control

One targeted attack is found. The attack is HTTP POST request with a specific HTTP body. The HTTP body is a json format data that follows the format of the real reply of the Philips Hue bridge. One example is shown in Figure 5.2. The instruction of Philips Hue API says that to control the hue device, a POST request with a json body and a valid URL should be send to the Philips Hue bridge. We can see that this attack is simulating a request to change the value of the Philips Hue bridge. This is a very targeted fuzzing that the attacker has already got knowledge of this implementation (knowing it is about Philips Hue) and Philips Hue control method.

5.5.2. Attack with the body following the *multipart/form-data* format

This is an attack that executed through HTTP POST with a specific body content. As is shown in Figure 5.3 and 5.4, the body content of the HTTP POST are following very similar patterns. The pattern is "`-------------{16_chars}\r \n Content-Disposition: form-data; name=\"on\"\r\n\r\ntrue\r\n-------------{16_chars}\r\nContent-Disposition: form-data; name=\"productid\"\r\n\r\n{random_payload}\r\n-------------{16_chars}-\r\n`", where {16_chars} is a random 16-long string contains only low-case letters and digits. {random_payload} is something random, where "botlight" fills with "%0000" and "000modscan" fills with nothing. Another user agent "mass" is also giving POTS requests that follow this pattern, whose {random_payload} is filled with an extremely long repeated (9944 times in one payload) "%A/telnet" or "%A/xmpp" or "%A/upnp". This is a very large payload that is obvious a fuzzing.

The URL of this attack is also following a pattern that is containing keyword (philips, hue, wemo, belkin, device, token, etc.) and a random 32-long string.

A conclusion of this attack could be get that this is a targeted fuzzing attack that is looking for the smart devices, xmpp, upnp or telnet services.

5.5.3. Attack with url

This kind of attack is through HTTP GET with URLs that follow a specific pattern:

1. /api/philips/hue/{32_chars}
2. /api/phi/light/{32_chars}
3. /api/philips1/hue/{32_chars}
4. /api/philips2/hue-link/{32_chars}
5. /api/belkin/wemo/{32_chars}
6. /api/tplink/light/{32_chars}
7. /api/hue/{0-750}
8. /api/phi/light/{32_chars}/tokens

9. /api/{32_chars}/tokens
10. /api/{32_chars}

Where {32_chars} means a 32-long string that contains low-case letters and digits randomly. And {0-750} means a number ranges from 0 to 750. All of these requests are targeted scanning attacks.

5.5.4. General Scanning Tools or Libraries

A number of scanning tools and libraries are found by the honeypot data. The scanning tools or libraries here means the ones that are not targeting to the smart devices.

- skipfish [18]
- Nikto [21]
- Jorgee: there is no much introduction about this on the Internet. But from the behavior of the request we can name this attack as Jorgee, which is a web scanner. Source code is not found. Behavior has been described in the previous section.
- masscan: This is also found in the user agent list. The source code address is also included in the Header. [60]
- Python library: urllib [9]
- /http:/testp3.pospr.waw.pl/testproxy.php: The IP of the proxy in your network can be shown.
- Proxyradar: On *https://proxyradar.com/* you can find open proxies.

5.5.5. Other unrelated attacks

In section 5.4.5, the number 2 and 3 describes two kind of attacks that is executing commands to find something but is not related to the IoT platform. More attacks could be found in the Table A.2 in the Appendix.

6 Conclusions and Future Work

In this thesis, a literature review on the IoT platforms including communication protocols and related security problems has been performed. Based on the result of the literature review, a design of an IoT platform honeypot (ThingPot) has been proposed. Moreover, a Proof of Concept (PoC) of ThingPot is implemented through the implementation of a IoT platform use case of Philips Hue (smart lights system). After implementing the PoC of ThingPot, data analysis based on the data generated by the ThingPot PoC has been performed.

6.1. Summary of Findings

Based on the literature review, XMPP is pointed as a flexible real time communication protocols that is feasible to be implemented in IoT platforms. It provides advantages with regards to the level of device management, which can be more structured and with additional layers that could impact on the security in a positive way (i.e. authentication, control, synchronization).

Currently, IoT platforms involve different types of protocols with different characteristics and functions. Instead of deploying an IoT platform with protocols focused on endpoints applications (e.g. telnet, simple web admin, ssh, etc.), XMPP can be used for the integration of different components in such a way they can be managed using a proper protocol focused on multi-node communication. Thus, with XMPP correctly implemented in an IoT platform, the structure of a heterogeneous network is simplified and it makes device management processes easier. In terms of security of an IoT platform, XMPP may provide additional layers which can also implement security mechanism (e.g. client-side authentication, device groups control, limitation of exposure of public services or backends, among others). This, could help to decrease the chances for attackers to find or reach the devices within the IoT platform.

Devices in different protocols and services could be linked to XMPP and thus there is no necessity to expose those devices directly to the Internet. And with a proper configuration of the XMPP implementation (including XMPP server authentication, account management, etc.), XMPP could improve the security in a easier way and increase the complexity of reaching devices for the attacker.

Data analysis of ThingPot PoC of XMPP part shows that attacker activities (even generic scans or requests) are very limited. In fact, there were no direct requests against the devices through the XMPP path. This supports the statement that XMPP increases the complexity for the attackers to reach the IoT platform and devices. Other reasons of this could also be that currently the attackers are not yet showing interests on the flaws of XMPP on an IoT platform. Or the attacks to XMPP are mainly on XMPP server side, which the ThingPot PoC has not covered. These reasons are still need to be discussed. However, security problems of XMPP's integration with IoT platform still need to be taken into consideration. More work on the protection of the XMPP accounts (prevention from being exposed, additional authentication) should be done to make a more secure IoT Platform with XMPP.

Smart devices are easy to be exposed and found on the Internet. For example, on *Shodan.io* a lot of Philips Hue could be found by simply searching for *"bridgeid"*.

Data analysis of ThingPot PoC of REST part shows that the IoT platforms and devices have been noticed by the attackers. Related attacks have been found on the ThingPot PoC. After analysis, five main kind of attacks are summarized in section 5.5. To conclude the attack patterns and methodology, firstly, attackers are looking for devices like Philips Hue, Belkin Wemo, TPlink, etc. Secondly, attackers are interested in getting the information of the smart devices and taking control of the devices. Thirdly, the attackers are using TOR network to mask their real source. The methodology that the attackers prefer is first a general scanning to look for opening services. And then a more targeted and specific attack is done by brute force the token or fuzzing. Moreover, ways to attack the Philips Hue are found according to the study of Philips Hue.

6.2. Contributions

Research questions are answered:

A literature review of the related protocols used by IoT devices is done. XMPP is found to be used to manage the whole IoT Platform as a connector between the devices and the users. Some of the existing vulnerabilities and the nature of known attacks on IoT protocols are found and explained. Thus the first two research questions are answered.

A well-designed functional prototype of an IoT Platform honeypot is given to answer the question "How can Honeypot technologies be implemented in order to be accurate, efficient and be able to identify security trends on IoT devices?". As listed in section 3.2.3, currently, the IoT honeypots are only focusing on single protocols including Telnet (Telnet IoT honeypot [55], IoTPOT [53], Multi-purpose IoT honeypot [39]), SSH (Multi-purpose IoT honeypot [39]), MQTT (Dionaea [24] [57]), ZigBee (ZigBee Honeypot [25]) and CWMP (HoneyThing [49]). Only single protocols in a single "device" is simulated. While the integration between protocols and the IoT platform is not addressed. ThingPot is the first IoT honeypot that is focused on the interaction between XMPP and REST as integrated IoT platform.

To answer the question "Are honeypot technologies suitable to identify security issues on IoT platforms?", a PoC of the ThingPot is implemented. By only running the ThingPot PoC for 46 days on 4 nodes (3 for REST API and 1 for XMPP), it has already caught a large amount of valuable data. By performing proper data analysis, we can already find and summarize attackers' activities against the IoT Platform. Thus ThingPot has shown the ability to identify security issues on the IoT Platforms. This proves that honeypot technologies are suitable to identify security issues on IoT platforms.

Based on the findings mentioned before, there are ways to prevent attacks on IoT Platforms:

- Encryption of the critical payload of the HTTP REST could be encrypted. The authentication method should be enhanced. For example, the whitelist of the Philips Hue bridge should be encrypted to avoid being stolen by the attackers.
- Moreover, instead of plain text, if the SSDP notify packages are encrypted, the location of the bridge will not be exposed.
- On the XMPP side, although XMPP makes the IoT platform communication network work in a more structured way and no attackers' activities are found in the ThingPot PoC, it is still essential to pay attention to the XMPP security. Protection of the accounts information is necessary.

Additionally, the dataset generated by the honeypot can be used to create IDS signatures (e.g. for SNORT or Suricata engines) in order to detect similar attacks against Philips Hue platforms and other generic REST-based scans against IoT devices.

6.3. Future Work

Three main opportunities for further work are identified in this section:

1. A self-built XMPP server helps to make the honeypot more effective.
   Building an XMPP server by ourselves gives more flexibility on managing the honeypot platform. By configuring or modifying the XMPP server, the attacks that are targeting the XMPP server or start from the XMPP server side could be caught and analyzed.

2. More exposures of the honeypot could make the honeypot more attractive and thus catch more data. The more accesses the honeypot has, the higher possibility an attacker can reach the honeypot, thus more information the honeypot could get. Increasing the number of public IP addresses used for the API part of the honeypot and the XMPP accounts help to increase the possibility for the attacker to reach the API. Moreover, not only increasing the number of honeypot can increase the data, but also exposing the "IoT platform" on public websites, forums helps.

3. More protocols (e.g. MQTT, UPnP) could be added to this IoT platform honeypot.

4. Dionaea module: The implementation of this research might be deployed either as a new module of Dionaea.

Bibliography

[1] Cve details. http://www.cvedetails.com/.

[2] Dream green house. http://dreamgreenhouse.com/projects/2013/xmpp/index.php.

[3] Json editor online. http://jsoneditoronline.org/.

[4] Computer security resource center-national vulnerability database. https://nvd.nist.gov/.

[5] Afrinic. http://www.afrinic.net/.

[6] Apnic. https://www.apnic.net.

[7] American registry for internet numbers. https://www.arin.net/.

[8] lacnic. http://www.lacnic.net/.

[9] 20.5. urllib — open arbitrary resources by url. https://docs.python.org/2/library/urllib.html.

[10] Ripe network coordination centre. https://www.ripe.net/.

[11] Tor project: Anonymity online. https://www.torproject.org/.

[12] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[13] Abdullah Al Hasib and MA Mottalib. Vulnerability analysis and protection schemes of universal plug and play protocol. In *Computational Science and Engineering (CSE), 2010 IEEE 13th International Conference on*, pages 222–228. IEEE, 2010.

[14] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22(7):97–114, 2009.

[15] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[16] Carsten Bormann, Angelo P Castellani, and Zach Shelby. Coap: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012.

[17] Martina Brachmann, O Garcia-Mochon, Sye-Loong Keoh, and Sandeep S Kumar. Security considerations around end-to-end security in the ip-based internet of things. In *Workshop on Smart Object Security, in conjunction with IETF83, Paris, France, March 23, 2012*, 2012.

[18] Bill Brenner. Attackers use skipfish to target financial sites.

[19] Thomas Brewster. How attackers can and will exploit upnp flaws. http://www.silicon.co.uk/workspace/how-attackers-will-exploit-upnp-105868.

[20] Brent N Chun, Jason Lee, Hakim Weatherspoon, and Brent N Chun. Netbait: a distributed worm detection service. *Intel Research Berkeley Technical Report IRB-TR-03*, 33, 2003.

[21] CIRT.net. Nikto2.

[22] Darknet. Xmppploit – a tool to attack xmpp connections. http://www.darknet.org.uk/2012/08/xmppploit-a-tool-to-attack-xmpp-connections/.

[23] N Dhanjani. Hacking lightbulbs: Security evaluation of the philips hue personal wireless lighting system, 2013.

[24] DinoTools. dionaea - catches bugs. `https://github.com/DinoTools/dionaea/blob/master/README.md`.

[25] S. Dowling, M. Schukat, and H. Melvin. A zigbee honeypot to assess iot cyberattack behaviour. In *2017 28th Irish Signals and Systems Conference (ISSC)*, pages 1–6, June 2017. doi: 10.1109/ISSC.2017.7983603.

[26] Joel L Fernandes, Ivo C Lopes, Joel JPC Rodrigues, and Sana Ullah. Performance evaluation of restful web services and amqp protocol. In *Ubiquitous and Future Networks (ICUFN), 2013 Fifth International Conference on*, pages 810–815. IEEE, 2013.

[27] Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures.* University of California, Irvine Doctoral dissertation, 2000.

[28] Gang Gan, Zeyong Lu, and Jun Jiang. Internet of things security analysis. In *Internet Technology and Applications (iTAP), 2011 International Conference on*, pages 1–4. IEEE, 2011.

[29] Oscar Garcia-Morchon, Sandeep Kumar, Rene Struik, Sye Keoh, and Rene Hummen. Security considerations in the ip-based internet of things. *draft-garcia-core-security-04*, 2013.

[30] Matthew Garrett. A quick look at the ikea trådfri lighting platform. `http://mjg59.dreamwidth.org/47803.html`.

[31] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva. Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials*, 17(3):1294–1312, 2015.

[32] Silva Granjal, Monteiro. Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials*, 17:1294 – 1312, 2015. doi: 10.1109/COMST.2015.2388550.

[33] John Hawkinson and Tony Bates. Guidelines for creation, selection, and registration of an autonomous system (as). 1996.

[34] Tom Hegel. Observing large-scale router exploit attempts.

[35] HIVEMQ. Mqtt 101 – how to get started with the lightweight iot protocol. `http://www.hivemq.com/blog/how-to-get-started-with-mqtt,`.

[36] HIVEMQ. Mqtt essentials part 2: Publish and subscribe. `http://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe,`.

[37] Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, and Jesus Alonso-Zarate. A survey on application layer protocols for the internet of things. *Transaction on IoT and Cloud Computing*, 3(1):11–17, 2015. URL `http://icas-pub.org/ojs/index.php/ticc/article/view/47`.

[38] Khan S. U. Zaheer R. & Khan S. Khan, R. Future internet: the internet of things architecture, possible applications and key challenges. *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260, 2012. doi: 10.1109/FIT.2012.53.

[39] P Krishnaprasad. *Capturing attacks on IoT devices with a multi-purpose IoT honeypot.* PhD thesis, INDIAN INSTITUTE OF TECHNOLOGY KANPUR, 2017.

[40] RAY LAPENA. More than 90with iiot in 2017. `https://www.tripwire.com/state-of-security/featured/90-pros-expect-attacks-risk-vulnerability-iiot-2017/`.

[41] Dangerous Link. Attempt to access to phpmyadmin/scripts/setup.php.

[42] C. Lu. Overview of security and privacy issues in the internet of things. *Internet of Things (IoT): A vision, Architectural Elements, and Future Directions.*

[43] Linda Markowsky and George Markowsky. Scanning for vulnerable devices in the internet of things. In *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on*, volume 1, pages 463–467. IEEE, 2015.

[44] Alexey Melnikov and K Zeilenga. Simple authentication and security layer (sasl). Technical report, 2006.

[45] Michael Mimoso. Upnp trouble puts devices behind firewall at risk. `https://threatpost.com/upnp-trouble-puts-devices-behind-firewall-at-risk/114493/`.

[46] LILY HAY NEWMAN. The web-shaking mirai botnet is splintering—but also evolving. `https://www.wired.com/2016/11/web-shaking-mirai-botnet-splintering-also-evolving/`.

[47] Kim Thuat Nguyen, Maryline Laurent, and Nouha Oualha. Survey on secure communication protocols for the internet of things. *Ad Hoc Networks*, 32:17–31, 2015. doi: http://dx.doi.org/10.1016/j.adhoc.2015.01.006.

[48] Im-Geol Oh and Jong-Il Lee. A study for vulnerability of security of the upnp home-networking. *Journal of the Korea Industrial Information Systems Research*, 12(2):30–36, 2007.

[49] omererdem. Honeything.

[50] Object Manage. Group (OMG). Data distribution services specification, v1.2. `http://www.omg.org/spec/DDS/1.2/`, 2015.

[51] Github Michel Oosterhof. Kippo, ssh medium-interaction honeypot. `https://github.com/desaster/kippo/`.

[52] Ed. P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. Technical report, RFC Editor, 2004.

[53] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. Iotpot: A novel honeypot for revealing current iot threats. *Journal of Information Processing*, 24(3):522–533, 2016.

[54] Philips. Configuration api. `https://www.developers.meethue.com/documentation/configuration-api`.

[55] Phype. Telnet iot honeypot.

[56] Gil Press. 6 hot internet of things (iot) security technologies. `https://www.forbes.com/sites/gilpress/2017/03/20/6-hot-internet-of-things-iot-security-technologies/#30b5e8081b49`.

[57] Carnivore Project. Dionaea honeypot. `http://dionaea.readthedocs.io/en/latest/index.html`.

[58] Niels Provos et al. A virtual honeypot framework. In *USENIX Security Symposium*, volume 173, pages 1–14, 2004.

[59] Wind River. Security in the internet of things. *whitepaper*, 2014.

[60] robertdavidgraham. Masscan: Mass ip port scanner.

[61] Tara Salman. Internet of things protocols and standards. `http://www.cse.wustl.edu/~jain/cse570-15/ftp/iot_prot/`.

[62] Zach Shelby, Klaus Hartke, and Carsten Bormann. The constrained application protocol (coap). 2014.

[63] Zhengguo Sheng, Shusen Yang, Yifan Yu, Athanasios Vasilakos, Julie Mccann, and Kin Leung. A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities. *IEEE Wireless Communications*, 20(6):91–98, 2013.

[64] Meena Singh, MA Rajan, VL Shivraj, and P Balamuralidhar. Secure mqtt for internet of things (iot). In *Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on*, pages 746–751. IEEE, 2015.

[65] sqmk. Finding philips hue bridge on network. `https://github.com/sqmk/Phue/wiki/Finding-Philips-Hue-bridge-on-network`.

[66] ITU Strategy and Policy Unit (SPU). The internet of things-executive summary. 2005. doi: http://www.itu.int/osg/spu/publications/internetofthings/InternetofThings_summary.pdf.

[67] studioimaginaire. A python library for the philips hue system. `https://github.com/studioimaginaire/phue`.

[68] Symantec. Web attack: Jorgee vulnerability scanner.

[69] Dinesh Thangavel, Xiaoping Ma, Alvin Valera, Hwee-Xian Tan, and Colin Keng-Yan Tan. Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE, 2014.

[70] Yong Wang, Garhan Attebury, and Byrav Ramamurthy. A survey of security issues in wireless sensor networks. *IEEE Communications Surveys & Tutorials*, 2006.

[71] XMPP-IOT. Python read and write. `http://www.xmpp-iot.org/tutorials/python-tutorial/`, .

[72] XMPP-IOT. Being friend with philips hue lights. `http://www.xmpp-iot.org/tutorials/python-philiphshue/`, .

[73] Quan-Feng YAN, Yue-Bin WANG, Yan XU, Li-Qiang LIU, and Bo FANG. Design of smart home system based on service encapsulation and device abstraction. 2015.

A

Count Remark	URL
31607	/api/
2984	/api/list/
2910	/
622	http://testp3.pospr.waw.pl/testproxy.php
280	/http:/84.19.176.29:80/mysql/admin/
275	/http:/84.19.176.29:80/sql/php-myadmin/
274	/http:/84.19.176.29:80/phpmyadmin/
274	/http:/84.19.176.29:80/PMA2012/
274	/http:/84.19.176.29:80/pma2015/
273	/http:/84.19.176.29:80/PMA2015/
273	/http:/84.19.176.29:80/PMA2014/

Table A.1: Top 11 URLs from HTTP requests taken from the proxy logs

Count	Request	Remark
31	"\x00\x00\x00B\xA4"	
8	"OPTIONS / RTSP/1.0"	not targeted
8	"Gh0st\xAD\x00\x00\x00\xE0\x00\x00\x00x\x9CKS“\x98\xC3\xC0\xC0\xC0\x06 \xC4\x8C@\xBCQ\x96\x81\x81\x09H\x07\xA7\x16\x95e&\xA7 * \x04$&g + \x182\x94\xF6\xB000\xAC\xA8rc\x00\x01\x11\xA0\x82\x1F\x5C‘&\x83\xC7K7 \x86\x19\xE5n\x0C9\x95n\x0C;\x84\x0F3\xAC\xE8sch\xA8\`xCF4′J\x97\xA9\x82 \xE30\xC3\x91h]&\x90\xF8\xCE\x97S\xCBA4L?2 = \xE1\xC4\x92\x86\x0B@\xF5‘ \x0CT\x1F\xAE\xAF]"	
8	"\x04\x01\x00P\xC0c\xF660\x00"	
8	"OPTIONS / HTTP/1.0"	
7	"USER test +iw test :Test Wuz Here"	
5	"@\x00\x00\x00\x00Z\xE6\xCA\x22BX\x1C\x86\x9B\xB0\xB2N\xF6(\x13o\xF5 \xB03Xo\xC0.\xA6W\xB0d\xD7Qsf\xD1\xA1\x0C\xB0V\xC32.\xB3@\x 0BI\x01\x00\xF6\x867\x9A + \x13\x02I!?[\x9D = \x08\xE4N"	
5	"FOO /FOO-sfi9876 HTTP/1.1"	
4	""	
4	"\x05\x02\x00\x02"	

Table A.2: Top 10 non HTTP requests taken from the proxy logs

Count	IP	TOR?
1338	104.223.123.98	Yes
1060	192.42.116.16	Yes
889	89.234.157.254	Yes
736	79.172.193.32	Yes
717	204.85.191.30	Yes
694	94.242.246.24	Yes
671	94.242.246.23	Yes
669	91.223.82.156	Yes
663	5.254.79.66	Yes
649	78.109.23.1	Yes
610	176.126.252.11	Yes
572	216.218.222.13	Yes
565	163.172.212.115	Yes
564	185.170.42.4	Yes
555	196.54.55.13	No
550	109.163.234.9	Yes
532	193.90.12.88	Yes
521	185.170.41.8	Yes
499	79.137.67.116	Yes
479	163.172.67.180	Yes

Table A.3: Top 20 source IP addresses from all requests taken from the api logs

Count	IP
20	193.70.95.180
18	164.52.7.132
15	220.181.159.73
12	47.203.93.185
8	66.240.205.34
7	93.115.95.207
6	180.97.106.162
5	110.90.95.163
4	185.100.87.246
4	208.100.26.229
3	72.11.140.74

Table A.4: Top 10 IP address of the non HTTP requests taken from the proxy logs

Count	autonomous system number (ASN)
24	36351
23	4134
23	12876
23	16276
17	28573
13	15169
9	60781
9	4837
9	14061
8	62567
8	63949
8	3223
7	8100
7	23650
6	23724
6	10439
6	16509
6	36375
6	42570
6	3462

Table A.5: Top 20 countries that IP addresses is located

Count	Regional Internet Registry (RIR)	Remark
258	ripencc	Réseaux IP Européens Network Coordination Centre (RIPE NCC) [10] for Europe, Russia, the Middle East, and Central Asia
142	arin	American Registry for Internet Numbers (ARIN) [7] for the United States, Canada, several parts of the Caribbean region, and Antarctica.
110	apnic	Asia-Pacific Network Information Centre (APNIC) [6] for Asia, Australia, New Zealand, and neighboring countries
31	lacnic	Latin America and Caribbean Network Information Centre (LACNIC) [8] for Latin America and parts of the Caribbean region
16	afrinic	African Network Information Center (AFRINIC) [5] for Africa
3		no RIR is recorded

Table A.6: The RIR of the IP addresses

Count	User Agent
1328204	"-"
31567	"shooter"
20162	"Mozilla/5.0 SF/2.10b"
2984	"botlight"
2378	"000modscan"
1867	"httpget"
831	"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"
622	"Mozilla/5.0 (Windows NT 5.1; rv:32.0) Gecko/20100101 Firefox/31.0"
607	"0000modscan"
313	"Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko"

Table A.7: Top 20 user agents from HTTP requests taken from the proxy logs

Count	User Agent
225	"-"
47	"Mozilla/5.0 SF/2.10b"
1	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8) AppleWebKit/536.5 (KHTML, like Gecko) Chrome/19.0.1084.56 Safari/536.5"
1	"Mozilla/5.0 (compatible; MSIE 8.0; MSIE 9.0; Windows NT 6.0; Trident/4.0; InfoPath.1; SV1; .NET CLR 3.8.36217; WOW64; en-US)"
1	"curl/7.53.1"
1	"Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"

Table A.8: User Agents of the non HTTP requests taken from the proxy logs

Count	User Agent
1327485	"-"
31567	"shooter"
2984	"botlight"
2378	"000modscan"
1867	"httpget"
606	"0000modscan"
154	"Mozilla/5.00 (Nikto/2.1.5) (Evasions:None) (Test:map_codes)"
107	"Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome /59.0.3071.115 Safari/537.36"
98	"Mozilla/5.0 SF/2.10b"
85	"curl/7.52.1"

Table A.9: Top 10 User Agents from the requests started with "/api" taken from the proxy logs

Count	User Agent
20064	"Mozilla/5.0 SF/2.10b"
831	"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; rv:11.0) like Gecko"
719	"-"
622	"Mozilla/5.0 (Windows NT 5.1; rv:32.0) Gecko/20100101 Firefox/31.0"
313	"Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko"
275	"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:47.0) Gecko/20100101 Firefox/47.0"
236	"Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.0.2) Gecko/2008092809 Gentoo Firefox/3.0.2"
186	"Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko"
158	"Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; Touch; rv:11.0) like Gecko"
103	"Mozilla/5.0 (Windows NT 10.0; WOW64; Trident/7.0; Touch; LCJB; rv:11.0) like Gecko"

Table A.10: Top 10 user agents from URLs not started with "/api" taken from the proxy logs

Count	Referer	Remarks
20	https://proxyradar.com/	It has a list of working open proxies.
13	http://84.19.177.29:80	
12	http://84.19.176.29/phpmyadmin	
11	http://84.19.176.29:80	
10	http://vegenis.servequake.com/	Valid URL to reach the node of the honeypot.
6	http://84.19.177.29/phpmyadmin	
5	http://84.19.177.29/	
5	http://morris.jusanet.org/api/belkin/wemo/af3bdcebd800931357951db376e0dad7	This is a very specific URL that is targetting to Belkin Wemo.
4	http://84.19.176.29/	Valid URL to reach the node of the honeypot.
4	http://morris.jusanet.org/	Valid URL to reach the node of the honeypot.
3	http://84.19.177.29/api/	Valid URL to reach the node of the honeypot.
2	http://84.19.177.29:80/	Valid URL to reach the node of the honeypot.
2	http://www.google.com/search?hl=ru&q=free+proxy+checker&sourceid=navclient-ff&ie=UTF-8	
2	http://www.google.com/search	
1	http://84.19.176.29/api/	Valid URL to reach the node of the honeypot.
1	http://84.19.177.29/phpmyadmin/index.php	
1	http://84.19.177.29/db/phpMyAdmin-3/	
1	http://84.19.176.29:80/	Valid URL to reach the node of the honeypot.
1	http://84.19.176.29:80/shell?%65%63%68%6F%20%63%61%6E%6C%61%6E%67	It could be decoded as "http://84.19.176.29:80/shell?echo can-lang" This URL is reported on the *www.abuseipdb.com.*
1	http://84.19.177.29/pma/	

Table A.11: Referers taken from the API logs