POLITECNICO DI TORINO

Dipartimento di Elettronica e Telecomunicazioni

Corso di Laurea Magistrale in Communications and Computer Networks Engineering

Tesi di Laurea Magistrale

Next Generation Honeypot for IoT

Supervisors:

Prof. Marco MELLIA

Prof. Danilo GIORDANO

Candidate:

Eustachio CANCELLIERE

S278031

December 2021

Contents

List of Tables

List of Figures

Abstract

The exponential growth of the number of IoT devices in the last years will increase the number of attacks perpetrated against them, since more IoT devices will be used to carry out illegal activities on the Internet. An example may be Mirai, a botnet composed of 200.000 and 300.000 infected device with peaks of 600.000 infected devices, mainly routers and cameras. In 2016 it was able to launch DDoS attacks against Krebs on Security and Deutsche Telekom leaving 900.000 Germans offline. This poses a great threat for all the systems and infrastructures connected to Internet since they may be of vital importance for human daily activities. Nowadays many solutions have been developed to identify threaths from the Internet such as Firewalls, Honeypots, Intrustion Detection Systems. On the other hand, given the limited number of Honeypot solutions for IoT devices, we present a methodology to build a digital twin of an IoT device obtained through the interaction with the physical device and then use it as honeypot. A digital twin gives a logical representation of a physical device and is able to emulate its internal state. We developed a software to automatically parse the packets captured during the interaction between a user and an IoT device, extract statistics and store the packet fields in a database. This database represents the dictionary of requests and responses used to build the digital twin of a given device. We applied this methodology on a testbed environment composed of an heterogeneous set of popular IoT home network devices connected through the Internet via a router instrumented with our tool. We also tried to perform man in the middle in order to increase the amount of plain text traffic exchanged from each device using mitmproxy with no success, since no app trusted the mitmproxy self signed certificate. We performed some analysis on the captured traffic for all the devices and we found out that all of them exchanged traffic mainly with respect to remote hosts than directly with their companion app although they were in the same LAN network. Moreover they contacted a handful of remote ports for both TCP and UDP protocols. The analysis of the TLS version used showed that although it is considered a recommendation to not support prior version than TLSv1.2, the vast majority of devices were still using TLSv1.0 during the Client Hello handshake, whereas the servers presented most of the times at least a TLSv1.2 version in Server Hello handshake. A few of them supported also TLSv1.0 and TLSv1.3. In order to have a proof of the effectiveness of this methodology we have built a first digital twin based on a Philips Hue Bridge that we have used as honeypot. We exposed this honeypot on the Internet and we collected traffic from August to mid November. Moreover it was recognized as an IoT device by Shodan. We then compared the traffic received from this honeypot with respect to other two honeypots that are Tanner and CameraObcura running in the same months. Tanner is a generic honeynet that emulates basic web sites running on port 80 whereas CameraObscura is impersonating a D-Link camera. The analysis of the CVEs discovered in the requests sent from remote hosts revealed that our honeypot was able to detect more exploits directed against IoT devices such as cameras or home routers. Moreover the amplification path of the common requests between is grater than 1 on average with respect to Tanner. None of the requests performed against HuePot were APIs belonging to a Philps Hue bridge. We speculated that remote hosts could be more interested to HuePot and CameraObscura than Tanner because their percentage of growth of the paths is much evident than Tanner.

Chapter 1

Introduction

The Internet of Things defines a network of connected devices with low capability in storage and computing, like sensors, actuators and more in general, embedded devices. Because of this lack of resources it is not possible to adopt high level security mechanisms, hence they are more vulnerable and more likely to be successfully exploited by adversaries. On the other hand this type of devices are everywhere in our everyday lives, from home network devices like routers, connected cameras, smart speakers, smart cars, to infrastructures and industrial environment. According to Cisco Annual Internet Report [1] in 2018 the amount of devices connected to Internet was 18.4 billion and the amount of IoT devices accounted for 33%, 6.1 billion. By 2023 instead the amount of IoT devices will be around 14.7 billion, 50% of the overall amount of networked devices which will be around 30 billion. This poses a great threat if we consider that already in 2018 SonicWall [2] was able to record 32.7 million IoT malware attacks globally. In the first semester of 2019 Kaspersky [3] detected more than 100 millions attacks against their honeypots, whereas in the first half of 2018 they detected around 12 millions threats. In the first half of 2021 Kaspersky [4] detected 1.5 billion attacks against IoT devices, 58% of them was carried out through Telnet protocol. The attacks conducted against IoT tried to take control of the devices through the automatic exploitation of known and yet not fixed vulnerabilities that could be one listed on some kind of database like National Vulnerability Database[1] or a zero-day vulnerability. Another important threat is represented by botnets, a set of devices that have been infected by a malware and can be controlled by a command and control in a centralized way or through a P2P decentralized model. A botnet can be used for many purposes, like DDoS attacks, mining, or can also be bought by other cybercriminals. A well known botnet is Mirai, which appeared for the first time in late 2016 and according to Antonakakis et al. [5] it was able to reach a steady state population in the range between 200.000 and 300.000 infections with a peak of 600.000 infected devices, mainly routers and cameras, by bruteforcing Telnet and SSH login before of collapsing at 100.000 infections by the end of February 2017. With this amount of devices it was able to launch DDoS attacks against Krebs on Security [6] with an astonishing 600 Gbps volume of traffic and Deutsche Telekom [7] among the others. During the years, after the release of Mirai source code [8] many botnet are born from it, like Gafgyt, Hajime and the number of Mirai like botnets is still growing. In order to keep up with respect to this quickly evolving environment it is necessary to always be up to date with respect to the newest threats that may compromise a network. On the other hand, if a network is under attack, we would like that such attack would be deflected against some physical or virtual device that can be compromised without harming the devices that we would like to protect. Both goals can be accomplished through the implementation and deployment of a Honeypot. In this thesis we present a methodology based on the creation of a database of requests and responses exchanged by a set of IoT devices in order to build a digital twin for each of them. In this way it is possible to reflect with a given level of accuracy the state of a physical device. As a consequence of

[1]https://nvd.nist.gov/

this, it is possible to build a honeypot based on the digital twin of a real device in such a way that we avoid to expose directly a physical device on Internet with the risk of being compromised. Moreover we avoid to contact IoT devices publicly available on the Internet to get the responses for a request that is not present in the database.

1.1 Research questions

In the thesis we want to understand what type of features can we extract from the traffic generated by IoT devices, and whether by exposing a honeypot built upon the digital twin of a real IoT device we are able to achieve better performances in terms of volume of traffic received and interest from remote hosts than a generic honeypot. More in detail we try to answer to the following questions:

- What type of features are we able to extract from the analysis of the traffic generated toward a wide group of IoT devices?
- What type of differences are present in the traffic collected by a digital twin honeypot with respect to generic honeypots or IoT ones built differently?
- Are we able to detect whether a remote scanner was interested in our honeypot by analyzing the requests received?

We show that there are some common features in the traffic exchanged by IoT devices that could be useful in the detection phase of such devices connected to a network. Moreover we show that the honeypot built with this method is able to achieve globally better performances in terms of volume of traffic received from remote scanners with respect to Tanner and CameraObscura. An initial analysis on the paths received from HuePot showed that it received more requests related to IoT devices than the other two honeypots. In the end the from analysis of the percentage of growth of each path received by HuePot and CameraObscura computed on a daily basis showed that the paths received only by HuePot had an higher growth than those received only from CameraObscura, probably because HuePot was recognized as an IoT devices by an higher number of scanners than CameraObscura and Tanner.

1.2 Methodology

The adopted methodology is composed of different steps reported in figure 1.1.

1. Firstly we develop a software to perform packet parsing and dictionary (or database, from now on we will use both therms interchangeably) creation automatically. In this way we can use a traffic sniffer tool and the pcap fields can be automatically inserted in a database of packets captured per device.

2. The second step was to study the IoT devices present in the lab collecting as much informations as possible about the features they implement through researches on Internet and active scanning. Then we applied the aforementioned methodology and we performed some further analysis about the traffic collected.

3. In the third step we select a set of the packets stored in the database related to a Philips Hue and we use them to set up a first honeypot named HuePot.

4. In the last step we compare the traffic collected from the previous honeypot with respect to the traffic collected from another generic honeypot called Tanner[2] and CameraObscura[3] that simulates the behaviour of a D-Link DCS-2530L camera.

[2]https://github.com/mushorg/tanner

[3]https://github.com/CMSecurity/CameraObscura

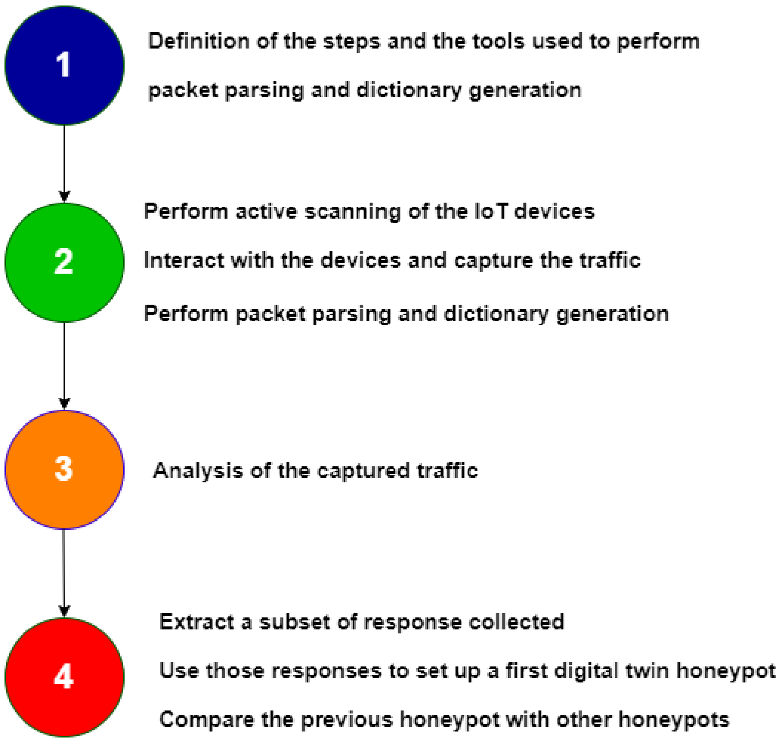

1
Definition of the steps and the tools used to perform packet parsing and dictionary generation
2
Perform active scanning of the IoT devices
Interact with the devices and capture the traffic
Perform packet parsing and dictionary generation
3
Analysis of the captured traffic
4
Extract a subset of response collected
Use those responses to set up a first digital twin honeypot
Compare the previous honeypot with other honeypots

Figure 1.1: Steps performed

1.3 Organization of the thesis

The next chapters are organised as follows: in Chapter 2 we present the honeypot taxonomy and the state of the art honeypots. In Chapter 3 we describe the laboratory testbed and the tools used to capture traffic, perform packet parsing and generate the dictionary of collected packets. In Chapter 4 we present the outcomes of the analysis performed on the traffic collected from the devices present in the laboratory. In Chapter 5 we present HuePot, a honeypot built using the methodology described in Chapter 3 and we compare the traffic gathered by HuePot with respect to the traffic collected by Tanner and CameraObscura. Finally in Chapter 6 we present the conclusions and the future works.

Chapter 2

State of the art

In this chapter we present a honeypot definition and the taxonomy associated to this concept. Then we describe some IoT honeypot features and we present some state of the art honeypots.

2.1 Honeypot taxonomy

A honeypot is a security tool composed of a computer system that is intended to attract cyberattacks, like a decoy. It mimics a target for hackers, and uses their intrusion attempts to gain information about cybercriminals and the way they are operating, or to distract them from other real targets. There exist many honeypots of different types, implementations and goals. Also, a set of honeypot can form a honeynet. This type of security mechanism has some benefits with respect to traditional Intrusion Detection Systems: in IDS systems the false negative frequency can be high and administrator may ignore the alarms generated by the system, in honeypot the frequency of false positives alarms triggered by unimportant events mistaken as attacks is by definition very low because they are deployed in darknets, and any interaction with them is only due to malicious traffic. On the other hand

the frequency of false negatives is also low for the same reason.
As reported by Javier Franco et al.[9] existing honeypots can be classified with respect to the set of parameters showed in 2.1. Among the other parameters, low interaction honeypots only emulate a set of protocols, without giving access to the operating system. This kind of honeypot is easy to be set up, less costly, with low risk of being compromised, hence less maintenance. On the other hand they can be easily detected by attackers because of those limitations. An example of low interaction honeypot is Honeyd [10], which is able to create a set virtual hosts on a network running different type of services like Telnet, Pop, Rpc, Snmp, with different types of operating systems. This last feature is based on the transport layer packet crafting. It works mainly as distraction to slow down potential attacks. High interaction honeypots can be obtained using real devices or virtual environment that emulates a device. On the other hand this comes at an higher cost in terms of resources because the risk of being compromised is higher and once compromised there is the need to rebuilt the system, hence this type of honeypot needs higher maintenance costs. An example of high interaction honeypot is Siphon [11], which is composed of a set of physical IoT devices like IP cameras, NVR, IP printer that are connected to the internet through a set of tunnels, in such a way that on the internet are present 85 real IoT devices in different geographical regions, hence reaching high scalability. Medium interaction honeypots provide a level of interaction in between a low interaction and high interaction honeypot emulating more services than a low interaction at an higher risk of being compromised, but making them more difficult to be detected. An example of medium interaction honeypot is ThingPot [12].
Another important parameter is the purpose of a honeypot: research honeypots are used to gather and analyze information about attacks in order to discover new attacks and to develop better protection against them. Production honeypots are

usually implemented to divert an attack from a valuable asset to a decoy, hence preventing the access to the actual system of the organization that implements it. The honeypot based on the methodology implemented in next chapters can be considered a medium high interaction honeypot since it emulates the status of a real device with low risk of being compromised because it does not expose a real device or a virtual environment, hence requiring less maintenance. The methodology applied guarantees an high level of scalability since it is possible to expose a number of digital twins proportional to the number of physical devices from which it is possible to extract traffic.

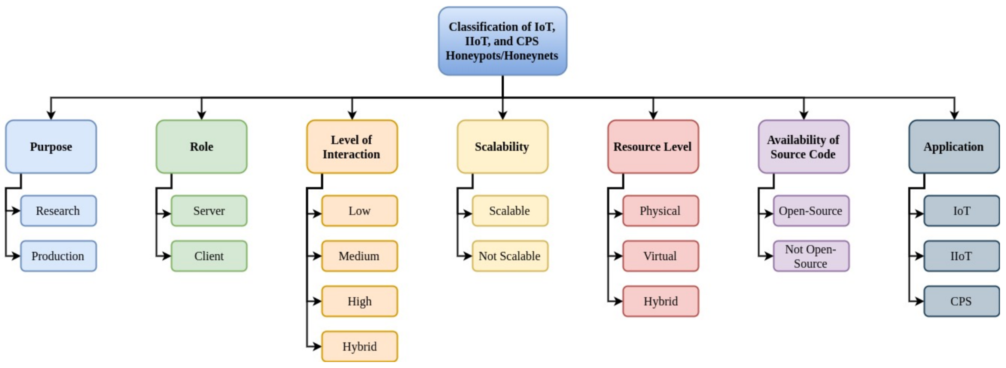


Figure 2.1: Honeypot taxonomy scheme described in [9]

2.2 IoT honeypot state of the art

2.2.1 IoTCandyJar

T.Luo et al. proposed IoTCandyJar [13], an intelligent interaction honeypot that is able to simulate an IoT device without the risk of the honeypot being compromised. Indeed the honeypot gathers the responses for the requests captured by sending those requests to IoT devices publicly available on Internet through a module called "IoT scanner". From the responses gathered they can build the behaviour of different devices. In this way there is no need to purchase IoT devices or to

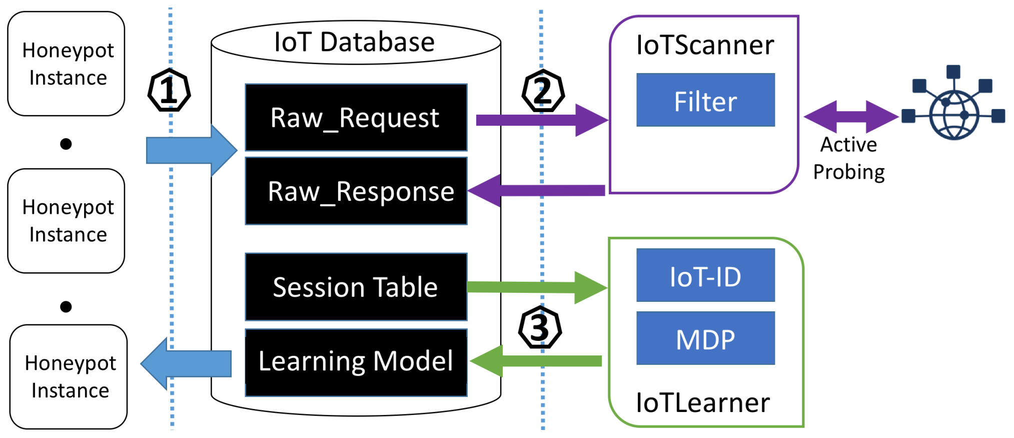


Figure 2.2: IoTCandyJar honeypot scheme described in [13]

virtualize them in order to build a high interaction honeypot. They also use a module called "IoT learner" to train a ML model based on Reinforcement Learning to learn which is the response that an attacker expect with highest probability in order to maximise the interaction with it, hence having a more realistic response. The drawback of this method is related to the usage of probes sent to real IoT devices in Internet that must not harm the devices, so it is necessary to take a proper filtering action before sending those probes.

2.2.2 ThingPot

Meng Wang et al. proposed ThingPot [12], a medium interaction honeypot able to simulate an entire IoT platform implemented with XMPP and a REST API to mimic the behaviour of a Philips Hue smart lighting system. The attacker could connect to the honeypot instances directly through the REST APIs or as XMPP client as well. Most of the captured requests were REST HTTP requests with respect to the XMPP part. From the logs it was possible to note that attackers were looking for devices like Philips Hue, Belkin Wemo, TPlink. Once they had found one, they would attack it to take control of the device through fuzzing or brute force. They also noted that often the attackers were using TOR network to hide their identities. Also a set of scanning tools were used to gather informations and left their fingerprints in the logs like Nikto[1], masscan[2] and others.

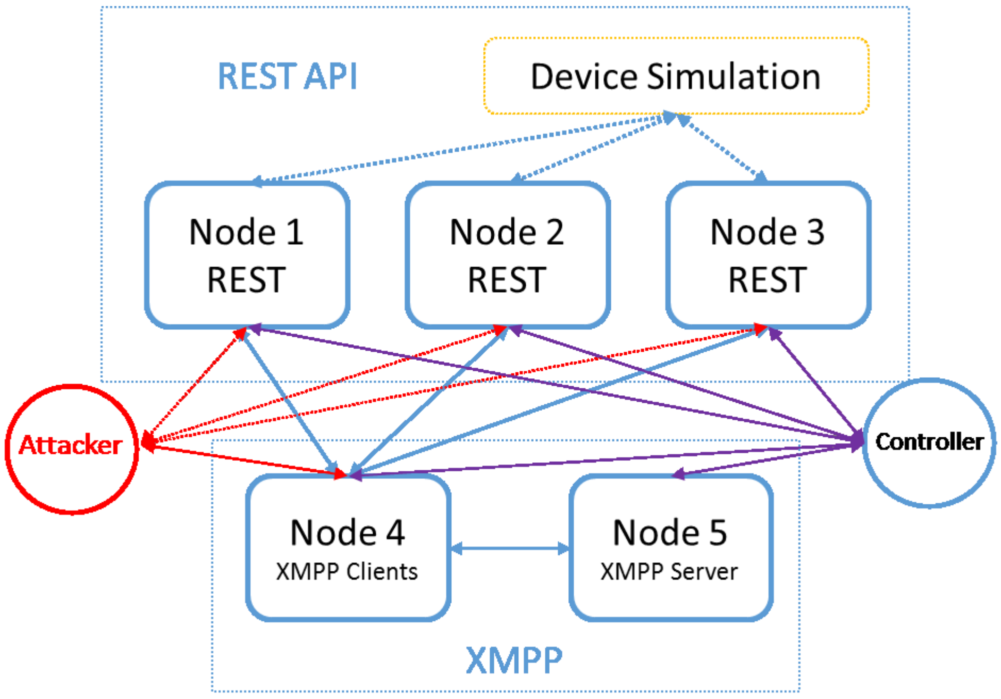


Figure 2.3: ThingPot honeypot scheme represented in [12]

[1]https://cirt.net/Nikto2

[2]https://github.com/robertdavidgraham/masscan

2.2.3 U-PoT

Muhammad A. Hakim et al. proposed U-PoT [14], a honeypot framework specific for IoT devices that uses UPnP protocol. This framework allows to automatically create a honeypot representation from the UPnP device description documents extracted from IoT devices, hence reaching high scalability. Its ability of emulating real devices has been tested considering four emulated devices, all of them were seen as real devices by OpenHAB[3].

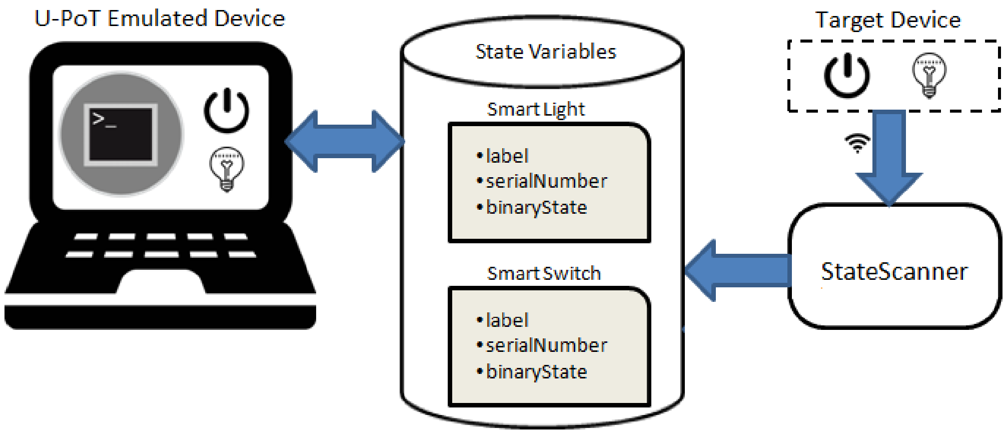


Figure 2.4: U-PoT honeypot scheme represented in [14]

[3]https://www.openhab.org/

Chapter 3

Methodology description

This chapter describes the methodology applied to capture, parse and store packets in order to build a dictionary where each entry corresponds to a packet captured that could be used to interact with an attacker. We describe and show the network setup used to connect the devices and we describe Nmap, the tool used to perform active scanning on the devices. Then we describe Tcpdump packet sniffer and the methodology applied to perform traffic capture. After that we describe the packet parser tool built using Scapy Python library. In the end we will discuss the database MongoDB choice and how entries have been stored in the dictionary.

3.1 Laboratory testbed description

All the sessions of packet capture have been performed using the IoT laboratory of Smartdata research group at Politecnico di Torino. The laboratory environment is based on the Mon(IoT)r Testbed[1] realized and deployed by Imperial College and used in the work presented by Ren et al. [15] about information exposure related

[1]https://moniotrlab.ccis.neu.edu/tools/

to IoT devices. This setup allows eventually the labs where this testbed is deployed to connect to each other through VPN to perform experiments using each other lab.

3.1.1 Mon(IoT)r Testbed environment

Mon(IoT)r testbed is a software that allows to create a controlled environment with either wired and wireless network connectivity provided to IoT devices, allowing to perform automatic collection of traffic related to devices connected to it. This traffic is then tagged automatically by MAC address or IP address. Moreover the testbed allows to define different virtual network interfaces used to run multiple experiments independently from each other in a controlled or uncontrolled environment. In the controlled network it is possible to run experiments with respect to a single device because it will be part of an isolated environment, whereas in an uncontrolled network environment the devices can see each other and can communicate. It is also possible to perform man-in-the-middle of HTTP and TLS connections. The testbed requires a set of software components that are Mitmproxy, Tcpdump in order to perform packet capture, ISC DHCP to automatically assign IP addresses and iptables to redirect traffic for MITMT purposes. There are also a set of software components developed by [15] that allow to control the testbed. For our tests we performed packet capture and MITM manually, whereas we used the network interfaces of the Mon(IoT)r testbed.

In the following lists is represented the set of network interfaces present in the tesbed, whereas in figure 3.1 is reported the testbed implementation:

- **eth0**: used to intercept all the traffic exchanged from/to Internet
- **eth-switch**: interface that hosts **switch-vlan10, switch-vlan11, switch-vlan12** virtual interfaces. The first is used for uncontrolled experiments, whereas the others are used for controlled experiments

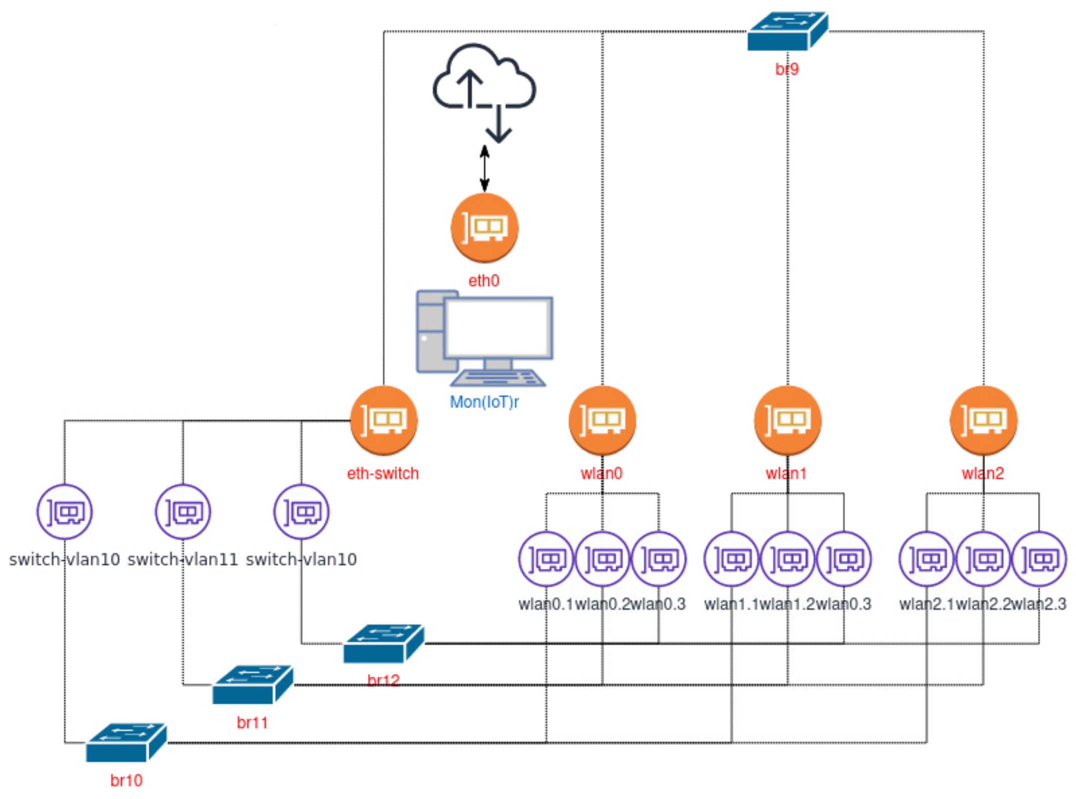


Figure 3.1: Mon(IoT)r testbed

- **eth-mirror**: interface used to perform port mirroring of the traffic running on the **eth-switch** switch interface. It hosts virtual interfaces **mirror-vlan10, mirror-vlan11, mirror-vlan12**, used to perform mirroring of vlan10,vlan11,vlan12.

- **wlan0**: Wi-Fi adapter on 2.4GHz network that hosts the virtual interfaces **wlan0.1, wlan0.2, wlan0.3**, used for unctontrolled experiments

- **wlan1**: Wi-Fi adapter on 5GHz network that hosts the virtual interfaces **wlan1.1, wlan1.2, wlan1.3**, used for controlled experiments

- **wlan2**: Wi-Fi adapter on 5GHz network that hosts the virtual interfaces **wlan2.1, wlan2.2, wlan2.3**, used for controlled experiments

Those interfaces are then bridged together in the following configuration:

- **br9**: used to bridge **eth-switch, wlan0, wlan1, wlan2** interfaces
- **br10**: used to bridge uncontrolled network traffic interfaces **switch-vlan10, wlan0.1, wlan1.1, wlan2.1**
- **br11**: used to bridge controlled network traffic interfaces **switch-vlan11, wlan0.2, wlan1.2, wlan2.2**
- **br12**: used to bridge controlled network traffic interfaces **switch-vlan12, wlan0.3, wlan1.3, wlan2.3**

3.2 Network scanning

Network scanning defines a set of techniques used to extract fingerprints from a network such as the number of hosts present in a network, whether they are reachable, what devices are present, the services they are running and possibly the presence of known vulnerabilities, in order to perform further analysis on the system based on the information extracted. Network scanning can be divided in two main categories that are active scanning and passive scanning.
Active scanning works by sending automatically crafted probes to a target host in order to extract the informations previously cited. In this way it is possible to interact with all the hosts present in a network crafting generic or targeted probes. In this category we can find network scanning, port scanning and vulnerability scanning. The drawbacks of this set of techniques is that they can become very intrusive and as a consequence there is the need to allocate bandwidth to this type of scanning procedures, hence they could damage the behaviour of the hosts present in a network. Another drawback is represented by the presence in a network of firewalls and IDS that could log the traffic generated by scanners as harmful and

as a consequence it would be dropped, hence the scanning would be ineffective. Passive scanning instead allows to extract information related to an host by sniffing the traffic that it generates without any type of interaction with the endpoints. Indeed there is no need of crafting probes and as a consequence of this, it is not intrusive. If there is a firewall in the system, it is more likely that a passive scanning would be successful than an active scanning. Moreover because it does not require bandwidth it can be deployed for long term analysis. On the other hand a drawback of this technique is that if a host in a network is silent and does not generate traffic, it will not be fingerprinted.

3.2.1 IoT device scanning

Also IoT devices can be fingerprinted using active or passive scanning with the goal of building some sort of classifier. This can be accomplished also because IoT devices performs very limited tasks and as a consequence of this their behaviour could be very predictable with respect to the behaviour of non IoT devices in terms of generated traffic. On the other hand it is important to choose the right features that must be extracted and that could be used to build a classifier. Sivanathan et al. [16] proposed a method to recognise an IoT device in a network by actively scanning its open ports using TCP probes with the aim of building a hierarchical classifier of TCP ports to scan in order to recognise the IoT devices running in a given network. Shahid et a.l [17] proposed a method to perform device recognition based on passive scanning of the network traffic and selecting as features the sizes of the first N packets received and sent, plus the first N-1 inter-arrival times between the first N packets received and sent. They were able to reach an accuracy of 99.9% related to what device has generated the traffic using a Random Forest Classifier.

3.2.2 Nmap scanning

Nmap (Network Mapper) is an open source, cross platform active scanning tool that has been used to detect the port status of the set of devices presented in table 4.1. It is used among the others, to perform host discovery, port scanning, service version and OS detection. It works using specially crafted packets, like the one described in [18]. Based on the type of response received and if it has been received one, it is possible to define the status of a port the service running on it and the OS. As reported on the Nmap manual[2], the status of a port can be:

- **Open**: there is a service listening on that port
- **Closed**: no service is listening on that port
- **Filtered**: a firewall blocks Nmap such that it cannot state if a port is open or close
- **Unfiltered**: there is no firewall blocking the port but Nmap cannot state if it is open or close

There are also other port status that are **open|filtered** or **closed|filtered**, which state that Nmap cannot distinguish whether a port is open or filtered or if it is closed or filtered respectively. There exist many port scanning techniques that Nmap implements that rely on UDP and TCP transport layer protocol. Some of them are the following:

- **TCP syn scan**: it is based on Nmap sending a SYN packet to the host under scanning. If it receives back a packet with RST flag the port is closed, if it receives back a packet with SYN/ACK flag the port is open whereas if it does not receive anything, the port is marked as filtered. It is very fast as it does

[2]https://nmap.org/book/man.html

not perform an entire TCP connection set up but on the other it requires root privileges on the attacking machine.

- **TCP connect scan**: it is based on setting up an entire TCP connection set up and as a consequence of this, it is slower than TCP SYN scan and it can be easily logged on receiver's machine. It can be used when a user does not have root privileges because it uses the connect system call that is also used by other processes.
- **UDP scans**: it sends UDP packets without payload to generic ports, whereas for well known ports running UDP it sends a specific payload. The UDP scanning is generaly slower than TCP scanning because open and filtered ports do not send back any response and Nmap must retransmit the packet before considering a port as open|filtered.
- **TCP NULL, FIN, and Xmas scans**: it is based on sending TCP packets setting the flag NULL, FIN or all the flags active.

3.3 Tcpdump capture methodology applied

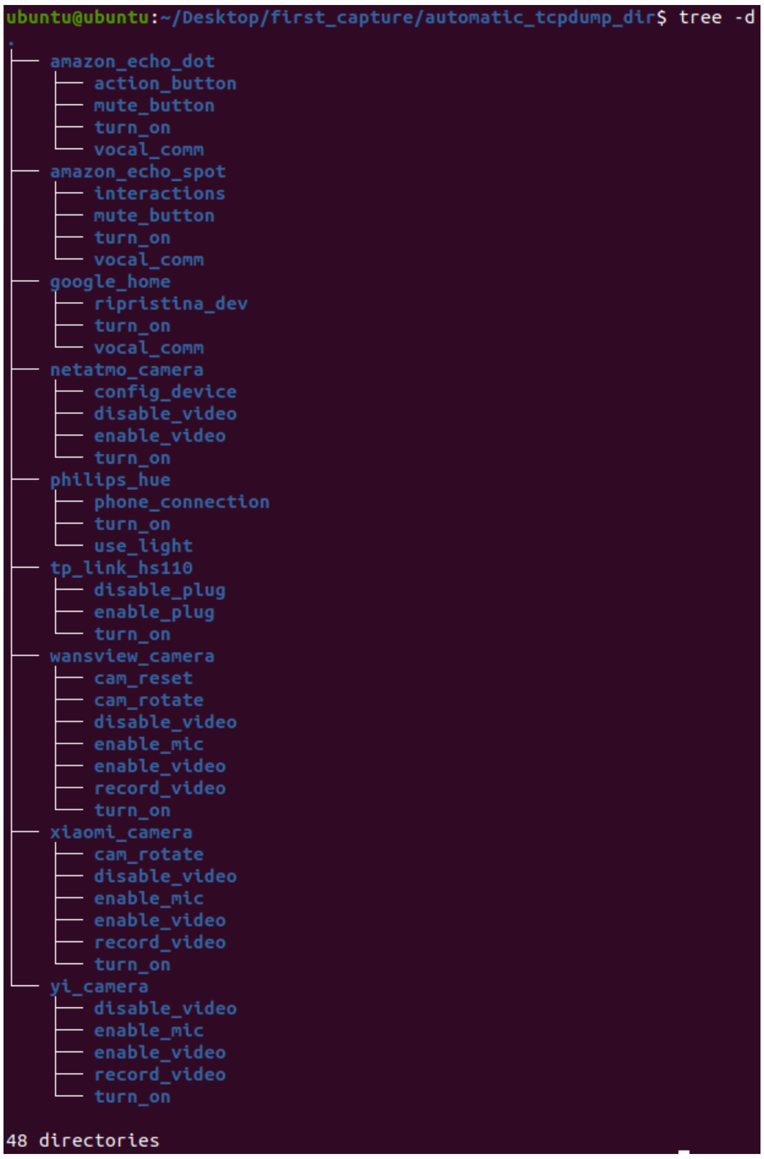
The packet capture has been performed using Tcpdump, a command line packet sniffing tool that can be used to analyze the behaviour of the hosts present in a network and the traffic generated by their application. It is a powerful tool that allows to capture traffic based on many protocols selecting a network interface. It also allows to perform packet filtering, as shown in 3.1. This traffic can be stored with pcap file extension for further analysis using other tools like Wireshark, since both are based on libpcap[3] library. The traffic captured was generated by a smartphone Android using the apps provided by each vendor. This choice is

[3]https://github.com/the-tcpdump-group/libpcap

sustained by the works proposed by Mauro Junior et al.[19] and [20] about the security flows found in the source code of the companion apps of IoT devices. Indeed they were able to find that Android app Kasa for Mobile companion for TP-Link smart plug used a custom encryption function Caesar cipher and an hardcoded seed to encrypt the communications in local network. They also were able to decrypt the traffic exchanged between a custom app build based on the vulnerabilities found in Kasa app and the smart plug. The capture operations have been performed using the options reported in code snippet 3.1. Each capture has been stored in a pcap file with a filename divided in device name, action performed and date. Each file then was saved in a tree of different folders as reported in figure 3.2. In this way it was possible to implement a wrapper in the packet parser program that has facilitated the pcap file parsing.

Listing 3.1: Example of capture performed with Tcpdump

```
sudo tcpdump host 192.168.10.109 -s0 -i eth-mirror -w
    philips_hue_turn_on_$(date +"%d_%m_%Y_%H_%M_%S").pcap
```



```
ubuntu@ubuntu:~/Desktop/first_capture/automatic_tcpdump_dir$ tree -d
.
├── amazon_echo_dot
│   ├── action_button
│   ├── mute_button
│   ├── turn_on
│   └── vocal_comm
├── amazon_echo_spot
│   ├── interactions
│   ├── mute_button
│   ├── turn_on
│   └── vocal_comm
├── google_home
│   ├── ripristina_dev
│   ├── turn_on
│   └── vocal_comm
├── netatmo_camera
│   ├── config_device
│   ├── disable_video
│   ├── enable_video
│   └── turn_on
├── philips_hue
│   ├── phone_connection
│   ├── turn_on
│   └── use_light
├── tp_link_hs110
│   ├── disable_plug
│   ├── enable_plug
│   └── turn_on
├── wansview_camera
│   ├── cam_reset
│   ├── cam_rotate
│   ├── disable_video
│   ├── enable_mic
│   ├── enable_video
│   ├── record_video
│   └── turn_on
├── xiaomi_camera
│   ├── cam_rotate
│   ├── disable_video
│   ├── enable_mic
│   ├── enable_video
│   ├── record_video
│   └── turn_on
└── yi_camera
    ├── disable_video
    ├── enable_mic
    ├── enable_video
    ├── record_video
    └── turn_on

48 directories
```

Figure 3.2: Pcap capture folder tree

Dev name	Action captured
Amazon Echo Spot	Turn on Press mute button Vocal commands Local interactions
Amazon Echo Dot	Turn on Press mute button Vocal commands Press action button
Google Home Mini	Turn on Vocal commands Factory reset
Philips Hue Bridge 2.0	Turn on Phone pairing User lights
Wansview Q5	Factory reset Camera rotate Disable video Enable microphone Enable video Record video Turn on
Yi Home Camera	Turn on Record video Enable video Enable microphone Disable video
Netatmo NSC01-EU 2.0	Turn on Enable video Disable video Phone pairing
Xiaomi Domo Camera	Turn on Record video Enable video Enable microphone Disable video Camera rotate
TP-Link HS110	Turn on Switch on Switch off

Table 3.1: Set of actions performed for each device

3.4 Automatic packet parsing with Scapy

The packet parsing operations have been performed using a program for packet manipulation and forging called Scapy[4], which is able to work with many protocols. It is also based on libpcap library, hence it can be used to read Tcpdump or Wireshark packets capture sessions as well. It can be used as shell or as a library to be included in other projects, in our case we used the latter solution in order to make the process of reading a pcap file, extracting the fields of each protocol layer and storing them in a database, as much automatic as possible. Other than Scapy the program uses other libraries like NFStream[5], Pymongo[6], Chardet[7]. The program is composed of the following elements:

- set of classes, one for each protocol, developed in order to store all the fields that we consider useful for a given protocol. The list of protocols extracted contains: Ethernet, IP, ICMP, ESP, TCP, UDP, HTTP, TLS.

- methods that are called in order to parse and extract the fields of a packet, generate metadata and try to detect the encoding of the payload using the aforementioned library Chardet. If Chardet is not able to decode a payload, it will be stored as a bytes object. Moreover during the packet extraction phase it will update the statistics related to each protocol stack layer.

- methods used to create an entry of the packet extracted in the dictionary using Pymongo library. In this way the fields extracted in previous phase are put together in a dictionary data structure that is passed to the database.

[4]https://scapy.net/
[5]https://www.nfstream.org/
[6]https://pypi.org/project/pymongo/
[7]https://pypi.org/project/chardet/

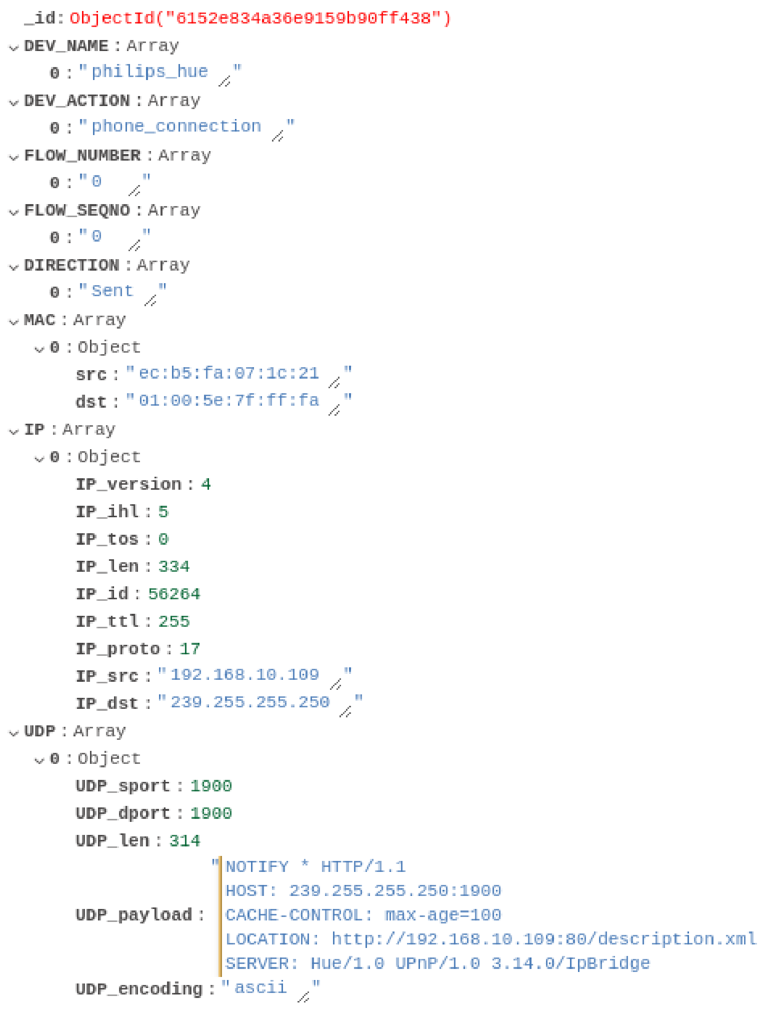
- methods used to compute and update statistics for each protocol layer such as the frequency of each protocol, overall number of packets exchanged, the ports used in transmission and reception for TCP and UDP. This set of statistics aggregated per device has been stored in a csv file for further analysis. Due to issues related to the statistics computed with this method, they have been recomputed using Tshark instead.
- methods used to compute and update statistics per flow using NFStream and ndpiReader, both are based on nDPI library. It has been used to extract for each flow the application name, application category and the requested server name. This set of statistics aggregated per device action has been stored in a csv file for further analysis.

3.4.1 Metadata collected

The set of metadata collected comprises information related to the device name, device action, number assigned to a packet belonging to a given flow, the encoding for a given payload retrieved by chardet, flow direction (whether a flow is received from a device or sent to), flow number associated to each flow per device action. All those parameters have been included in each entry of the dictionary in order to ease the answer process to a particular request sending the sorted packets belonging to a given flow.

3.5 Dictionary creation using MongoDB

After the packet parsing phase it was necessary to collect and store all the data as entries of a database of responses used against possible requests that the honeypots received. The choice of the DBMS to use was between a relational database such as MySQL and a non relational database such as MongoDB. The first database is

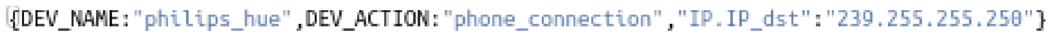


```
 _id: ObjectId("6152e834a36e9159b90ff438")
⌄DEV_NAME : Array
   0 : "philips_hue "
⌄DEV_ACTION : Array
   0 : "phone_connection "
⌄FLOW_NUMBER : Array
   0 : "0 "
⌄FLOW_SEQNO : Array
   0 : "0 "
⌄DIRECTION : Array
   0 : "Sent "
⌄MAC : Array
  ⌄0 : Object
     src : "ec:b5:fa:07:1c:21 "
     dst : "01:00:5e:7f:ff:fa "
⌄IP : Array
  ⌄0 : Object
     IP_version : 4
     IP_ihl : 5
     IP_tos : 0
     IP_len : 334
     IP_id : 56264
     IP_ttl : 255
     IP_proto : 17
     IP_src : "192.168.10.109 "
     IP_dst : "239.255.255.250 "
⌄UDP : Array
  ⌄0 : Object
     UDP_sport : 1900
     UDP_dport : 1900
     UDP_len : 314
                   "NOTIFY * HTTP/1.1
                    HOST: 239.255.255.250:1900
     UDP_payload :  CACHE-CONTROL: max-age=100
                    LOCATION: http://192.168.10.109:80/description.xml
                    SERVER: Hue/1.0 UPnP/1.0 3.14.0/IpBridge
     UDP_encoding : "ascii "
```

Figure 3.3: Example of Mongodb document

composed of a set of tables where a new entry is stored as a row. The new entry must match all the fields of that table in a fixed way. In order to retrieve some data from SQL like database it is necessary to perform a SQL query joining many tables in order to have a complete view of the required data. On the other hand, because this type of database has a fixed and static structure, it looses in terms of flexibility with respect to MongoDB. Indeed MongoDB is a NoSQL database, meaning that it does not use SQL in order to create and operate on a database. It is composed of a set of collections where each entry (the corresponding of a row in a MySQL database) is a JSON or BSON document. The flexibility of this model lies behind the possibility to store heterogeneous documents with different type of fields and structure of the information in the same collection, allowing to easily handle high volume of data with a complex structure in faster way than a MySQL database could, as shown by Győrödi et al. [21] and by Boicea et. al [22]. Moreover in MongoDB a query can be performed among the others through REST API, CLI if we are using the MongoDB shell or a specific set queries depending on the type of GUI or programming language we are dealing with, as reported in the MongoDB documentation[8].

Because of the heterogeneity and the high volume of the data collected, it was chosen MongoDB as database. The database obtained is composed of one collection that contains all the documents that have been generated by the packet parser program. Each document contains all the fields extracted from every packet plus the metadata described in section 3.2.1. The graphical representation of an entry is shown in figure 3.3 where it is possible to observe how in a document are stored many data type at the same time. Moreover the Transport and the Application layers can be different from packet to packet, increasing the heterogeneity in the data collected. An example of a query using MongoDB Compass is shown in figure 3.4, which query will select all the documents matching the device name, device action and the destination IP fields. An example of document is shown in figure 3.3.



```
{DEV_NAME:"philips_hue",DEV_ACTION:"phone_connection","IP.IP_dst":"239.255.255.250"}
```

Figure 3.4: Example MongoDB query using Compass

[8]https://docs.mongodb.com/manual/tutorial/query-documents/

Chapter 4

Analysis IoT devices and captured traffic

In this chapter describe the IoT devices on which the metodology described in the previous chapter was applied. We describe what data we were able to find looking for public available material on the Internet regarding each of them. Then we show the features extracted using Nmap and man in the middle tools like Mitmproxy. In the end we presented the statistics extracted from the traffic exchanged between devices and their companion app using the automatic packet parser described in the previous chapter, Tshark and Tstat, making comparisons with respect to other works on IoT network traffic analysis.

4.1 IoT devices footprinting

In our experiments are involved devices belonging to a wide set of different home network IoT categories, like smart speaker, IP camera, smart lighting, smart plug. For each category we selected the devices that an average consumer would buy, following the same methodology of [15]. In table 4.1 is presented the list of devices used and the category they belong to. In order to obtain as much information as possible we performed extensive research for each device looking for features that could be useful in further analysis and traffic captures, like the presence of APIs, whether a device allows local or cloud communications, the presence of known vulnerabilities that could be that could be exploited remotely. As it is possible to observe in table, all the devices use cloud communication and can be locally and remotely controlled through their companion app. For what concern the APIs available for each device, some of them are released by the vendor like Philips Hue, others are unofficials, like Google Home Mini. All of them have an authentication mechanism such as a token that can be obtained by connecting the user developer account to the API authorization url and that must be retrieved before of using the APIs, like Netatmo camera Oauth2 mechanism. Philips Hue instead implements a mechanism that allows to retrieve a token after having pressed the button present on the device. For what concern Wansview camera, the vendor did not release a documentation about the available APIs and also in this case the only APIs available are unofficial. All the IP cameras present in that list can also store video content locally on a micro SD, whereas Xiaomi Domo Camera allows to store recordings on the cloud. Regarding the known vulnerabilities found on the National Vulnerability Database, they account for vulnerabilities that can be remotely exploited without considering those that could lead to DDoS attacks. The vulnerabilities present in that column are related to information disclosure like those of Google Home Mini and Philips Hue, or command injection, like the one related to the Netatmo Camera. The CVE found were already fixed.

4.2 Devices active scanning

During the port scanning phase for each device we retrieved the port status, the OS running on each device and the service version of the process running on a given

Dev name	Dev type	LAN comm.	Cloud comm.	API avail.	Known CVE
Amazon Echo Spot	Smart speaker	NO	YES	Off. API	No CVE found
Amazon Echo Dot	Smart speaker	NO	YES	Off. API	No CVE found
Google Home Mini	Smart speaker	NO	YES	Unoff. API	CVE-2018-12716
Philips Hue Bridge 2.0	Bridge	YES	YES	Off. API	No CVE found
Wansview Q5	IP camera	YES	YES	Unoff. API	CVE-2012-3002
Yi Home Camera	IP camera	NO	YES	No API	No CVE found
Netatmo NSC01-EU 2.0	IP camera	YES	YES	Off API	CVE-2019-17101
Xiaomi Domo Camera	IP camera	YES	YES	No API	No CVE found
TP-Link HS110	Smart plug	YES	YES	Unoff. API	No CVE found

Table 4.1: IoT device features fetched from Internet

port. The final report is shown in table 4.2, where we have reported not only the open ports, but also the closed and the open|filtered ports for completeness reasons. What comes up is the very high number of open ports for the Google Home Mini that may be an issue if that device is not frequently updated because for each open port is present a service that could be affected by a given vulnerability. As a consequence of this it may provide many pathways for attackers to exploit the device starting from the services running on those open ports. Another issue is related to UDP ports: if one of those devices that have UDP open ports is exposed on the Internet it may act as a reflector for DDoS attacks, as reported in [23]. Indeed many IoT devices use protocols like SSDP, CoAP and SNMP that use UDP at transport layer. An example of them is Philips Hue Bridge that is using port 1900/UDP to run UPnP which has been detected as open|filtered by Nmap.

Device name	OS version detected	Port	Status	Service	Version service
Amazon Echo Spot	Linux 2.2.x-3.x	5000/udp	open|filtered	upnp	
		5353/udp	open|filtered	zeroconf	
		55442/tcp	open	nagios-nsca	
		55443/tcp	open	ssl/unknown	
Amazon Echo Dot	Linux 2.2.x-3.x	1080/tcp	open	socks5	
		8888/tcp	open	sun-answerbook	
		55442/tcp	open	nagios-nsca	
		55443/tcp	open	ssl/unknown	
Google Home Mini	Linux 3.1-3.10	8008/tcp	open	http	
		8009/tcp	open	ajp13	
		8012/tcp	open	unknown	
		8443/tcp	open	https-alt	OpenSSL 1.1.1j-dev 2.0.7 static
		9000/tcp	open	cslistener	
		100001/tcp	open	scp-config	
		10005/tcp	open	stel	
		10101/tcp	open	ezmeeting-2	
		5353/udp	open	zeroconf	
		68/udp	open|filtered	dhcpc	
		33434/tcp	closed	traceroute	
Philips Hue Bridge 2.0	Linux 2.2.x-3.x	80/tcp	open	http	nginx 1.x
		443/tcp	open	https	OpenSSL 1.1.1j-dev 2.0.7 static
		8080/tcp	open	http-proxy	
		5353/tcp	open	zeroconf	
		1900/udp	open|filtered	upnp	UPnP/1.0
Wansview Q5	Linux 3.1-3.10	80/tcp	open	http	lighttpd/1.4.52
		554/tcp	open	rtsp	AJSS/1.0.4
		65000/tcp	open	unknown	
		68/tcp	closed	sptx	
		68/udp	closed	dhcpc	
		40404/tcp	closed	dhcpc	
		39402/tcp	closed	htpdate	htpdate/1.1.30
		9999/tcp	open	abyss	
		5353/udp	open|filtered	zeroconf	
Yi Home Camera	Linux 2.2.x-3.x	5353/udp	open|filtered	zeroconf	
		4131/tcp	closed	stars	
Netatmo NSC01-EU 2.0	Linux 3.1-3.10	22/tcp	filtered	ssh	
		80/tcp	open	http	lighttpd/1.4.58
		5555/tcp	filtered	freeciv	
		5353/tcp	open|filtered	zeroconf	
		61374/tcp	open	unknown	
		5001/tcp	open	commplex-link	
Xiaomi Domo Camera	Not detected	5353/udp	open|filtered	zeroconf	
		68/udp	closed	dhcpc	
		32770/udp	closed	sometimes-rpc4	
TP-Link HS110	Linux 2.2.x-3.x (barebone)	9999/tcp	open	abyss	
		68/udp	open|filtered	dhcpc	
		1490/udp	open|filtered	insitu-conf	

Table 4.2: Nmap port scanning results

4.3 Mitmproxy against IoT devices

After the initial scan performed using Nmap we tried to intercept and replay HTTP and HTTPS messages exchanged between companion app and the IoT devices. We used a tool called "mitmproxy" [24] to perform this task. Unfortunately all the traffic exchanged between smartphone and devices were encrypted except for few messages. As a consequence of this we installed the mitmproxy certificate authority on the client Android smartphone but none of the IoT apps under test trusted the mitmproxy CA. This is in accordance with [15], where it is reported to not have performed MITM of TLS connections because it failed most of the time in their preliminary work and when successful it could affected devices behaviour. Indeed we should have performed one of the operations reported in [25] in order to correctly install a CA on client and IoT device, but those operations do not scale well with respect to the amount of devices we used because they needed some extra work depending on where the CA file had to be installed, hence they were considered out of the scope of this thesis.

4.4 Network traffic analysis

In this section we describe the statistics of the traffic captured aggregated per device. Firstly we present a set of global statistics, like the amount of bytes exchanged with UDP and TCP. Then we analyze the amount of remote hosts contacted and port numbers used to exchange traffic. In the end we describe some features related to the TLS protocol mechanisms implemented by the devices and the second level domain of the remote hosts contacted. This analysis can give an overview of the traffic generated by devices that can be used to build a mechanism of fingerprinting and classification of IoT devices based on their traffic peculiarities. Some devices send also API in clear text used for example to identify themselves, like Philips Hue, Netatmo camera, Wansview Camera and the Amazon Echo Spot. Other devices uses different ports, like Google Home mini that exchanges traffic on the 443/UDP. A peculiarity of Xiaomi camera is that it connects to the highest number of remote ports for UDP traffic. A difference between Amazon Echo Dot and Echo Spot is in the number of different ports that both devices use to communicate with remote hosts due to a different number of flows generated. For what concern TLS usage, the SSLv3 can be considered a peculiarity of a Wansview Camera with respect to the other devices, whereas the usage of TLSv1.3 can be considered a peculiarity of a Google Home Mini.

4.4.1 Global traffic statistics per device

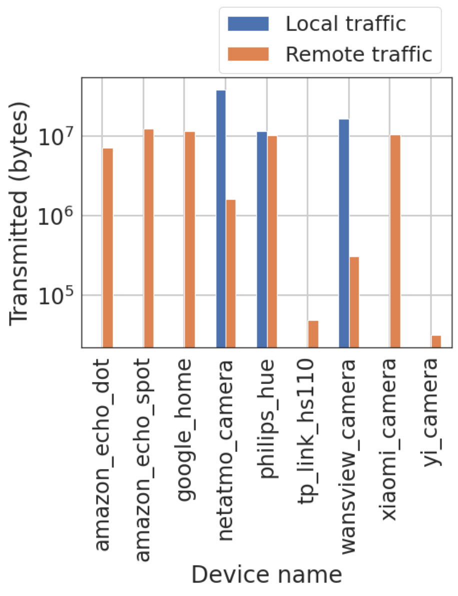


Figure 4.1: Number of bytes transmitted locally and remotely per device

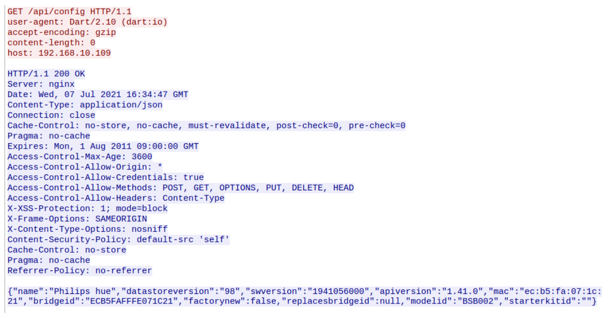
In figure 4.1 we show that all the devices exchanged traffic through the Internet in order to communicate with the smartphone. TP-Link and Yi camera have the lower number of bytes exchanged probably because the duration of the experiment was too short. Netatmo camera, Philips Hue and Wansview camera instead exchange a portion of traffic directly with the smartphone apps. Analysing the captured packets, we were able to extract a set of features related to the local traffic:

- Netatmo camera exchange locally only TCP and HTTP traffic in plain text. No encryption mechanisms were found in the local communications. An example of HTTP stream is reported in figure 4.2. This API has been exchanged during the connection procedure between IoT device and its companion app and it is sent periodically from the smartphone. After the set up of the communication, all the other api exchanged locally that have been captured have the same string f9f507e945c526717588d346ccef374a. The HTTP requests captured include files with .ts extension to the smartphone which requested them via GET method.

- Philips Hue Bridge exchanges traffic locally using SSDP, HTTP and TLS protocol. Indeed this device implements UPnP protocol based on SSDP to interact with other devices in the network. As a consequence of this it periodically sends broadcast discovery messages in order to declare its presence in the network, as reported in the documentation [26]. HTTP protocol instead is mainly used during connection phase between device and app, when the device sends data in json format about the device itself, as reported in figure 4.2. After that, all the successive communications between device and companion app are encrypted.

- Wansview camera transmits locally only UDP traffic related to real time communications toward its companion app. There is only one HTTP API sent in clear text captured during the connection phase between camera and app that has been reported in figure 4.10.

In figure 4.3 and 4.4 are reported the overall amount of UDP and TCP bytes in transmission and reception per device. The devices use TCP protocol mainly

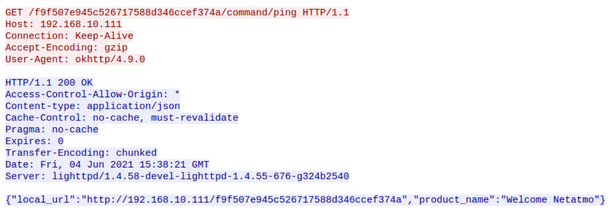


```
GET /api/config HTTP/1.1
user-agent: Dart/2.10 (dart:io)
accept-encoding: gzip
content-length: 0
host: 192.168.10.109

HTTP/1.1 200 OK
Server: nginx
Date: Wed, 07 Jul 2021 16:34:47 GMT
Content-Type: application/json
Connection: close
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Expires: Mon, 1 Aug 2011 09:00:00 GMT
Access-Control-Max-Age: 3600
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE, HEAD
Access-Control-Allow-Headers: Content-Type
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'
Cache-Control: no-store
Pragma: no-cache
Referrer-Policy: no-referrer

{"name":"Philips hue","datastoreversion":"98","swversion":"1941056000","apiversion":"1.41.0","mac":"ec:b5:fa:07:1c:
21","bridgeid":"ECB5FAFFFE071C21","factorynew":false,"replacesbridgeid":null,"modelid":"BSB002","starterkitid":""}
```

(a) Philips Hue Bridge API

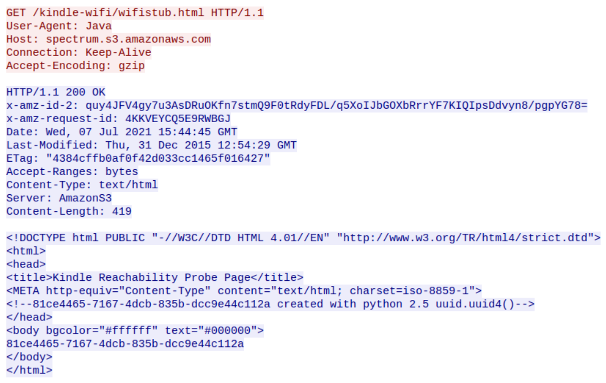


```
GET /f9f507e945c526717588d346ccef374a/command/ping HTTP/1.1
Host: 192.168.10.111
Connection: Keep-Alive
Accept-Encoding: gzip
User-Agent: okhttp/4.9.0

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-type: application/json
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Expires: 0
Transfer-Encoding: chunked
Date: Fri, 04 Jun 2021 15:38:21 GMT
Server: lighttpd/1.4.58-devel-lighttpd-1.4.55-676-g324b2540

{"local_url":"http://192.168.10.111/f9f507e945c526717588d346ccef374a","product_name":"Welcome Netatmo"}
```

(b) Netatmo Camera API



```
GET /kindle-wifi/wifistub.html HTTP/1.1
User-Agent: Java
Host: spectrum.s3.amazonaws.com
Connection: Keep-Alive
Accept-Encoding: gzip

HTTP/1.1 200 OK
x-amz-id-2: quy4JFV4gy7u3AsDRuOKfn7stmQ9F0tRdyFDL/q5XoIJbGOXbRrrYF7KIQIpsDdvyn8/pgpYG78=
x-amz-request-id: 4KKVEYCQ5E9RWBGJ
Date: Wed, 07 Jul 2021 15:44:45 GMT
Last-Modified: Thu, 31 Dec 2015 12:54:29 GMT
ETag: "4384cffb0af0f42d033cc1465f016427"
Accept-Ranges: bytes
Content-Type: text/html
Server: AmazonS3
Content-Length: 419

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Kindle Reachability Probe Page</title>
<META http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<!--81ce4465-7167-4dcb-835b-dcc9e44c112a created with python 2.5 uuid.uuid4()-->
</head>
<body bgcolor="#ffffff" text="#000000">
81ce4465-7167-4dcb-835b-dcc9e44c112a
</body>
</html>
```

(c) Amazon Echo Spot API

Figure 4.2: Example of plain text API exchanged between devices and their companion app

for HTTP and TLS traffic, whereas UDP traffic is mainly used for NTP and DNS data essential for the correct functioning of the devices. Netatmo camera uses ESP protocol based on UDP to communicate with remote hosts. From the figure we observe that the number of bytes exchanged using TCP protocol is higher than UDP for all the devices, except for Google Home, Wansview and Xiaomi camera, that instead use UDP packets also to communicate with remote hosts or directly with the smartphone.

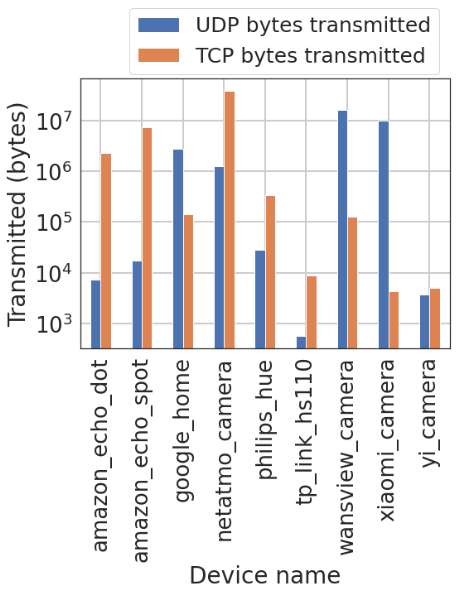


Figure 4.3: UDP TCP bytes transmitted by each device

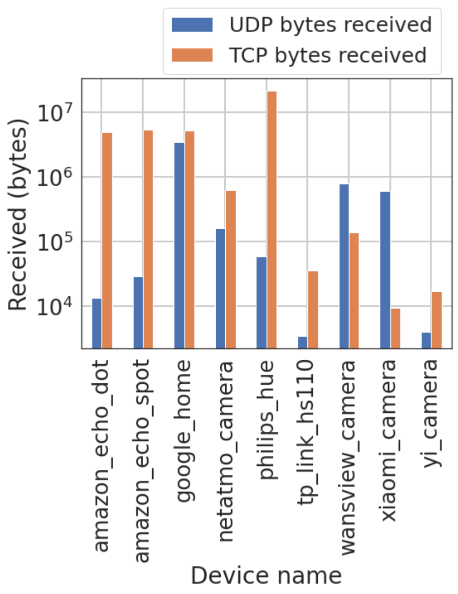


Figure 4.4: UDP TCP bytes received from each device

In figure 4.5 and 4.6 are represented the amount of bytes exchanged using HTTP and TLS protocols. It is evident that TLS traffic is way higher than HTTP exchanged traffic. Nevertheless there are a few plain text messages exchanged between devices and smartphone app or remotely for all the devices except for TP-Link smart plug and Yi camera for which no HTTP messages have been captured. Another example shown in figure 4.2 is related to Amazon Echo Spot, which periodically contacts a remote host using the api in figure to retrieve a Kindle Reachability Probe Page.

4.4.2 Analysis of the ports used

In figure 4.7 we show the number of distinct UDP and TCP remote ports that each device contacted. Those values are much smaller than the number of distinct TCP and UDP ports that each IoT device used to transmit and receive traffic

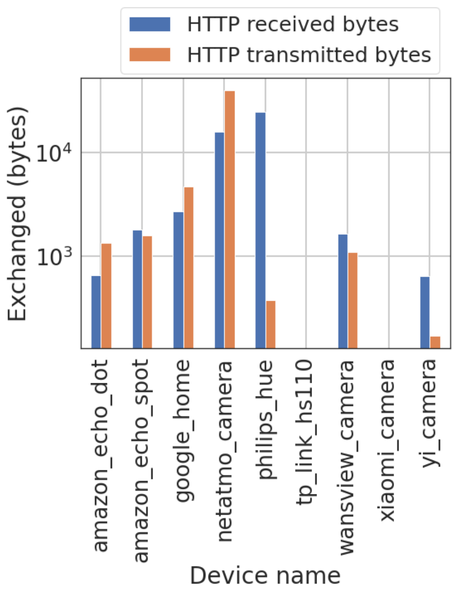


Figure 4.5: HTTP bytes exchanged

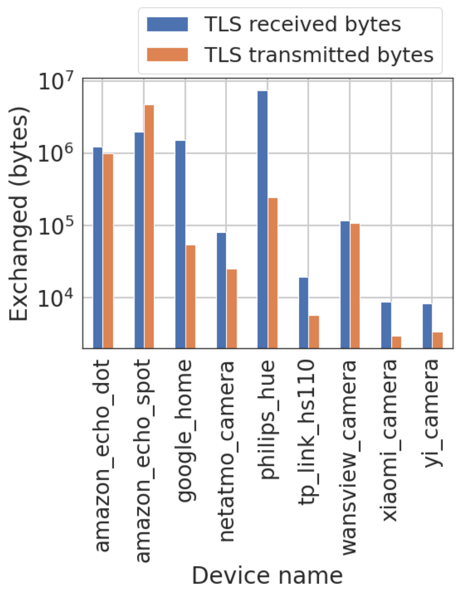


Figure 4.6: TLS bytes exchanged

except for TP-Link, Xiaomi Camera and Yi camera, as reported in figure 4.8. The traffic exchanged on UDP ports is mainly related to NTP and DNS protocols, since the devices connect most of the times to port 53/udp and 123/udp. Each device then differs from one to another with respect to other ports it uses: for example a peculiarity of Google Home Mini device is that it connects to port 443 UDP of remote hosts because it uses QUIC protocol. The remaining UDP ports are specific for each device and are used to exchange data. Xiaomi camera contacts the highest number of different UDP remote ports because it exchanges UDP data traffic not related to NTP or DNS protocols contacting different remote ports on different hosts. The TCP ports instead are used to exchange encrypted traffic on port 443/tcp or, in a much lower part, plain text traffic directed to port 80/tcp of remote hosts. The same behaviour has been found in the work of Mainuddin et al. Network Traffic Characteristics of IoT Devices in Smart Homes [27], in which they compared the number of remote ports and the number of different second level domains contacted by IoT devices and non-IoT devices. A similar work has been

carried out by Sivanathan et al. [28]. In both cases and in our tests IoT devices used to connect to a handful of remote ports for both TCP and UDP services and those numbers are smaller than the number of different remote ports contacted by non IoT devices.
On the other hand the number of different ports used from each IoT device is much higher than the number of open ports that the Nmap classified as open ports for both UDP and TCP protocols, as reported in figure 4.8. This behaviour can be explained with the fact that those devices may have a set of predefined rules for which they must block all the incoming traffic on a given port but they open a port only if they need to communicate. On the other hand Amazon Echo Spot and Amazon Echo Dot opened a different number of ports communicate with remote hosts although they share the same vendor: from the pcap files captured Echo Spot uses an higher number of TCP ports in transmission because it generates an higher number of flows than Echo Dot according to Tshark: the number of TCP flows for Echo Spot is 195, whereas the amount of TCP flows for Echo Dot is 103.

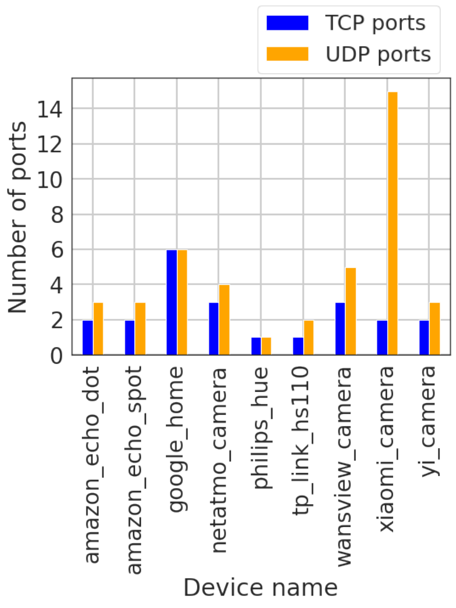


Figure 4.7: Number of remote ports contacted per device

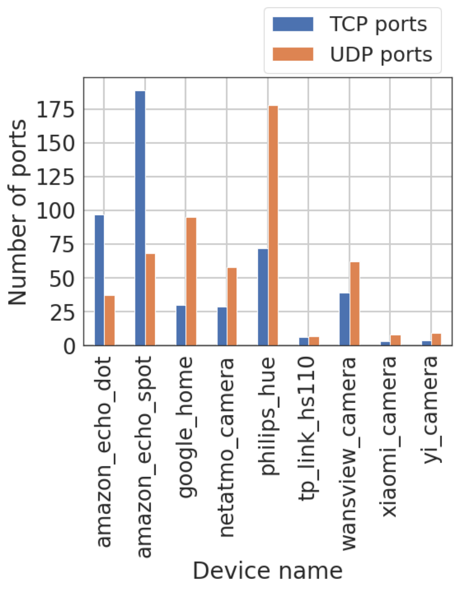


Figure 4.8: Number of ports used in reception by each device

4.4.3 Second level domains contacted per device

In table 4.3 are reported the second level DNS accessed by IoT devices obtained using Tstat[1]. As it is possible to observe, devices built by the same vendor share a similar set of domains, like Amazon Echo Spot and Amazon Echo Dot. It is also present the domain name meethue.com, which has been contacted by Echo Dot probably because of some interactions between Echo Dot and Philips Hue. Amazon Echo Dot contacted the domain mediaset.net when we required the news as a vocal command. In general the number of different remote domains contacted per device is limited to a handful of domains. This is in line with [28]: they found out that IoT devices tend to contact a smaller number of domains compared to non IoT devices.

[1]http://tstat.polito.it

Device Name	Domains
Amazon Echo Spot	amazon.com amazonvideo.com aiv-delivery.net googleapis.com amazonaws.com cloudfront.net amazoncrl.com amazon.co.uk amazonalexa.com amazon.it amcs-tachyon.com
Amazon Echo Dot	amazon.com cloudfront.net meethue.com mediaset.net fireoscaptiveportal.com theplatform.eu
Google Home Mini	googleapis.com google.com gstatic.com googleusercontent.com capital.it googlevideo.com
Philips Hue Bridge 2.0	Not found
Wansview Q5	ajcloud.net
Yi Home Camera	xiaoyi.com
Netatmo NSC01-EU 2.0	netatmo.net
Xiaomi Domo Camera	mi.com
TP-Link HS110	tplinkra.com tplinkcloud.com

Table 4.3: Second level domain resolved per device

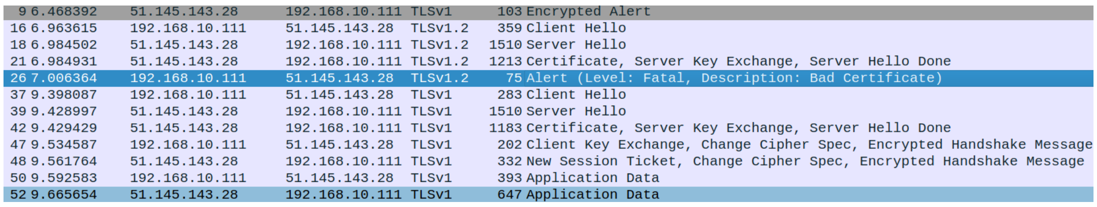
4.4.4 Analysis of TLS usage

In table 4.4 are reported for each device the TLS version advertised with the Client Hello message and established through Server Hello message as part of the TLS Handshake phase. The version established will be the highest version supported

Device Name	Version advertised	Version established
Amazon Echo Spot	TLSv1.0	TLSv1.2 TLSv1.0
Amazon Echo Dot	TLSv1.0	TLSv1.2
Google Home Mini	TLSv1.0	TLSv1.3
Philips Hue Bridge 2.0	TLSv1.0	TLSv1.0 TLSv1.2 TLSv1.3
Wansview Q5	TLSv1.2 SSLv3	TLSv1.2
Yi Home Camera	TLSv1.2	TLSv1.2
Netatmo NSC01-EU 2.0	TLSv1.0	TLSv1.2 TLSv1.0
Xiaomi Domo Camera	TLSv1.0	TLSv1.2
TP-Link HS110	TLSv1.2	TLSv1.2

Table 4.4: TLS version advertised and established per device

between client and server. As a consequence of this, the vast majority of the devices will set up connections using TLSv1.2, which is a non deprecated version. Nowadays SSLv3.0, TLSv1.0 and TLSv1.1 are considered deprecated whereas TLSv1.2 is the recommended version, as reported in [29], and [30]. Moreover IETF suggests to remove the support to versions older than TLSv1.2 in order to reduce the attack surface. The table in figure 4.4 instead reports that Client Hello messages uses TLSv1.0 most of the times and in the case of the Wansview Camera, also SSLv3. Moreover it is the only device to advertise two different TLS versions. This behaviour has been shown and analyzed also in the work IoTLS: Understanding TLS Usage in Consumer IoT Devices proposed by Paracha et al. [31]. According to that work, an IoT device may contain different TLS instances that different software can use independently from each other. For what concerns the established version, most of the devices uses at least TLSv1.2 version, for example Google Home Mini sets up connections using only TLSv1.3. There are some devices like the Amazon Echo Spot, Philips Hue Bridge 2.0, Netatmo NSC01-EU 2.0 that establish



```
 9 6.468392   51.145.143.28    192.168.10.111 TLSv1      103 Encrypted Alert
16 6.963615   192.168.10.111   51.145.143.28  TLSv1.2    359 Client Hello
18 6.984502   51.145.143.28    192.168.10.111 TLSv1.2   1510 Server Hello
21 6.984931   51.145.143.28    192.168.10.111 TLSv1.2   1213 Certificate, Server Key Exchange, Server Hello Done
26 7.006364   192.168.10.111   51.145.143.28  TLSv1.2     75 Alert (Level: Fatal, Description: Bad Certificate)
37 9.398087   192.168.10.111   51.145.143.28  TLSv1      283 Client Hello
39 9.428997   51.145.143.28    192.168.10.111 TLSv1     1510 Server Hello
42 9.429429   51.145.143.28    192.168.10.111 TLSv1     1183 Certificate, Server Key Exchange, Server Hello Done
47 9.534587   192.168.10.111   51.145.143.28  TLSv1      202 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
48 9.561764   51.145.143.28    192.168.10.111 TLSv1      332 New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
50 9.592583   192.168.10.111   51.145.143.28  TLSv1      393 Application Data
52 9.665654   51.145.143.28    192.168.10.111 TLSv1      647 Application Data
```

Figure 4.9: Netatmo camera TLS Bad Certificate

also connections using TLSv1.0 for different purposes.

- Philips Hue Bridge 2.0 establishes connections with a remote host which contacts the device when it was turned on and during the connection between device and companion app.

- Amazon Echo Spot establishes a connection with a remote host to which it sends a Client Hello with TLS version TLSv1.0. On the other side the remote host sends back a Server Hello with TLS version TLSv1.0 as well. For that remote host the maximum TLS version is TLSv1.0.

- Netatmo NSC01-EU 2.0 establishes connections with different remote hosts to which it sends a Client Hello with TLS version TLSv1.0 the remote hosts send back a Server Hello with TLS version TLSv1.0. This means that also for this remote host the highest TLS version supported is TLSv1.0. Another peculiarity of the traffic extracted is related to the handshake performed between the device and a remote host for which it is issued a Bad Certificate TLS alert message. After that message both endpoints try to perform the handshake one more time, but now the remote host present a TLSv1.0 version in Server Hello, whereas the camera uses a different port number. Once the connection has been teared down the camera uses TLSv1.0 version whereas the Server uses TLSv1.2 for further handshakes. In figure 4.9 is reported the entire set of handshake steps.

4.5 Wansview camera feature extraction

After the analysis of the global traffic we selected an Wansview Q5 camera to perform further analysis that could be useful to build the digital twin of an IoT device. The choice fell on this device because it had both port 80/tcp for HTTP protocol, 554/tcp for RTPS protocol open and also it was known the service version running on port 80/tcp. Moreover the vendor provides App for iOS, Android and Windows Phone, PC software for Windows and Mac, plus it is possible to control the camera through IE, Chrome and Firefox browsers. This camera is shipped with hardcoded default username and password that are **admin** and **123456** respectively. From the mobile app it was possible to extract the structure of the API related to the RTSP protocol communications, that is `rtsp://[username]:[password]@[ip-address]:554/live/ch0`. On the other hand the vendor does not provide any APIs, but because this camera can work with third party software like iSpy, Blueiris, IP Cam Viewer, we were able to find a set of API for many Wansview models, included ours. As a consequence of this we tried all of them, including also those APIs related to other cameras of the same vendor, but the vast majority of them didn't work except for the first two rows in table 4.5, that return a snapshot of the camera and that are specific of another model of camera which is Wansview Camera W6. Moreover they did not require any username and password to be used unlike the RTSP uri. Because of this we tried to run Nikto vulnerability scanner against it but with no success. No recent vulnerabilities have been found on NVD and on Exploit-db[2] databases related to this device. On Metasploit there were no exploits related to this camera, nor to this lighttpd version. The search for vulnerabilities on lighttpd/1.4.52 led to CVE-2019-11072, which has already been patched. The other exploits related

[2]https://www.exploit-db.com/

to lighttpd accounted for other versions. The last option was to sniff the traffic generated between the smartphone Wansview app and the camera. It was possible to extract another API using Tcpdump, which is sent in clear text and that carries a body in json format that includes an access token and an access key, as reported in figure 4.10. In red is represented the request sent with header and payload from the smartphone, whereas in blue is represented the response header and payload of the camera. All the other communications were UDP packets hence it was not possible to extract any other API.

Method	API
GET	/api/v1/snap.cgi?chn=0
GET	/api/v1/snap.cgi?chn=1
POST	/api/v1/lan-probe

Table 4.5: Wansview API

```
POST /api/v1/lan-probe HTTP/1.1
Accept-Language: it-IT,it;q=0.8
User-Agent: Mozilla/5.0 (Linux; U; Android 7.0; it-it; SM-G920F Build/
NRD90M) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0 Mobile
Safari/534.30
x-agent-token: f48b0e1c-cbdf-4bf3-bb41-e27ff0ca3a7f
Content-Type: application/json;charset=utf-8
Content-Length: 315
Host: 192.168.10.100
Connection: Keep-Alive
Accept-Encoding: gzip

{"meta":{"locale":"it","localtz":
60,"appName":"wansview","appVendorCode":"WVC","accessToken":"MmJlYTIwN
zAtZGRmOS0xMWU5LTllOGQtYjM0OGYxODE1NjY5Ljk0NGNjMTFjZWM5Nzg0Zjk5MjZlMjZ
hOWMyNDhkODE5LjE2MjI4MjIzMjg._rh9Cdkr5KWntRADPasqA9GFGCTjn2Mpvz0rPV6Dr
PA"},"data":
{"accessKey":"l7P5fB59Ny9Cg8V9RWCCv2f_85jr5RztK_UIsYVChSI"}}HTTP/1.1
200 OK
Content-Type: application/json
Content-Length: 78
Date: Fri, 04 Jun 2021 14:07:27 GMT
Server: lighttpd/1.4.52

{"status":"ok","code":0,"message":"","result":
{"deviceId":"WVC93C0IKE7TV2GN"}}
```

Figure 4.10: Wansview request and response with tokens and access key

4.6 Outcomes

The port scanning phase shows that some devices like Google Home Mini have many open ports, which may be a problem if the device is exposed on the Internet because those ports may represent a threat if the service version running on those ports is no up do date. Moreover perform man in the middle against IoT devices requires to adopt different strategies that may change from device to device and could also invalidate the correct behaviour of the devices. The analysis of the traffic exchanged between devices and their companion apps shows that TCP traffic is used to exchange HTTP and TLS packets, whereas UDP traffic is used to exchange traffic related to NTP and DNS. Apart from that, some devices, like Google Home, Wansview Camera and Xiaomi Camera use UDP also to exchange data and Philips Hue uses UPnP on port 1900/UDP. There is a trend in IoT devices for which they contact a lower number of remote ports and remote second level domains than a non-IoT device. This can be explained by the limited complexity of this type of devices because they only perform a few number of limited tasks. On the other hand the number of ports used to transmit traffic is much higher even though Nmap did not detect them. Moreover the TLS version advertised by IoT devices is lower than TLS version established by remote hosts probably because of lack of maintenance of the software installed on the devices. On the other hand remote hosts tend to use more updated TLS versions. The most important result that comes out from the analysis of the Wansview camera is that this type of devices are shipped with hardcoded username and password. If the credentials are not modified and the device is exposed on the Internet a remote host may take control of them. This is the answer to the first question of paragraph 1.1

Chapter 5

HuePot in the wild

In this Chapter we present HuePot, a first digital twin honeypot based on a Philips Hue Bridge. We presented also other two honeypots called Tanner and CameraObscura that we used as baseline to prove the effectiveness of the methodology applied. Firstly we made some comparisons of the global activities between the honeypots with respect to the volume of traffic captured, then we investigate the similarity between the set of remote hosts that contacted each honeypot and the frequency with which each path appeared in each system and the percentage of variation of each path over time. Thanks to the outcomes we answer to the last two questions of paragraph 1.1.

5.1 Honeypots implementation

In this section we describe the implementation details of each honeypot and the informations extracted from an initial analysis of the traffic collected by HuePot.

5.1.1 HuePot

HuePot was built starting from a subset of HTTP responses that we were able to intercept in clear text using Tcpdump. We have chosen to emulate this device because although the number of APIs intercepted is not big enough to have a complete description of it, many APIs are available on the Internet. Indeed we used them to increase the number of APIs that a remote host could request. After that we have set up a virtual machine running Ubuntu 20.04.2 LTS. Starting from the informations gathered through Nmap, we opened port 80/tcp and 8080/tcp and we run nginx/1.18.0 web server on the first port, whereas no services were installed on the latter. We did not open udp ports such as 1900/udp because of security reasons related to amplification DDoS attacks. On port 80/tcp we run the APIs and the banner grabbed on the Internet related to the device. HuePot collected traffic from 11th August to 10th November. The IP assigned was 130.192.167.247.

5.1.2 Tanner

Tanner is a honeynet composed of 16 IP addresses divided in two groups: a first octet of IPs has many open ports but not all of them run a service. A second octet of IPs instead has only port 80/tcp opened running a Nginx Web application server. Tanner is composed of two different components that are Tanner itself and Snare. Tanner[1] performs data analysis, classification and evaluation of the HTTP requests received. It also decides how Snare should answer to an attacker. Snare[2] on the other hand is able to convert generic web pages to attack surfaces that serves to the attacker. It was used as baseline with respect to HuePot and CameraObscura to compare them against a generic honeypot that uses as response a generic web

[1]https://github.com/mushorg/tanner
[2]https://github.com/mushorg/snare

page. Because of the presence of 16 different IPs, the comparisons between HuePot or CameraObscura and Tanner have always been performed considering only one IP from Tanner or averaging the data fetched from the two octects. When HuePot was set up, this honeynet was already running. The traffic collected by Tanner is referred to a period from August to the first days of October. Unfortunately it did not work continuously all the days of those months due to some issues present.

5.1.3 CameraObscura

CameraObscura[3] is a honeypot able to fake an IP Camera implementing features such as login attempts or firmware upload. We decided to fake a D-Link DCS-2530L IP camera because from HuePot logs we observed a request that was trying to exploit a vulnerability in that camera related to an unauthenticated endpoint that allows to get remote authentication and password disclosure. The request is **/config/getuser?index=0**. The goal is to compare the traffic collected by HuePot and another honeypot that is emulating an IoT device which does not expose the real web pages of the device in question. CameraObscura had only port 80 opened and collected traffic from 1st October to 10th November. The IP assigned was 130.192.167.246.

5.2 Analysis of the requests received by HuePot

In this paragraph we describe some peculiarities of the requests received by HuePot. This analysis could give us a first understanding about how remote hosts performed scans against HuePot. From the paths and the user agent of the requests received by HuePots it was possible to extract different scanning tools and organizations

[3]https://github.com/CMSecurity/CameraObscura

that sent traffic toward this honeypot The scanning tools observed are the following: Masscan[4], works like Nmap but is able to perform massive port scanning possibly over the entire Internet. Nuclei[5] is a vulnerability scanner which scans the hosts sending them requests forged starting from a template filled to exploit a given vulnerability. Colly[6] is a scraper. On the other hand the organizations found are reported in figure 5.1. Censys and Shodan are search engines that scan the Internet to return reports of the devices found. NetSystemResearch[7] is an independent organization that scans the Internet focusing on security topics such as IoT proliferation. LeakIX[8] is a platform that like Shodan indexes services and misconfigurations of devices found all over the Internet. Project Resonance[9] scans the Internet to study the security state of the Internet.
For what concerns the requests received by HuePot we have found many vulnerabilities that the remote hosts were trying to exploit. Some of them are reported in figure 5.1, where it is represented the number of times a requests has been received by the three honeypots and the percentage with respect to the total amount traffic received. Those vulnerabilities are related to embedded devices, mainly home routers and IP cameras. Remote code execution of an endpoint is the preponderant vulnerability they try to exploit. Many vendors have been found, such as D-Link and Netgear. From the table we can observe that some of those requests were not observed in Tanner and CameraObscura. From this analysis we can make an initial hypothesis that because HuePot was recognized as an IoT device, it may have received more different requests crafted to exploit IoT devices than the other

[4]https://github.com/robertdavidgraham/masscan
[5]https://github.com/projectdiscovery/nuclei
[6]https://github.com/gocolly/colly
[7]https://www.netsystemsresearch.com/
[8]https://www.leakix.net/
[9]https://github.com/redhuntlabs/Project-Resonance

two honeypots.

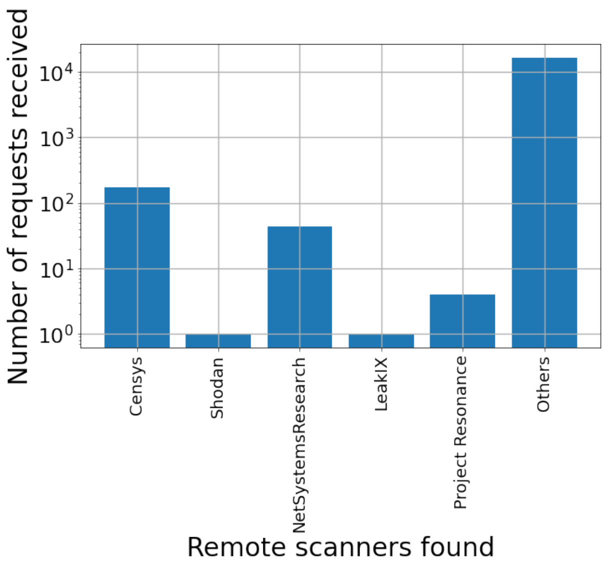


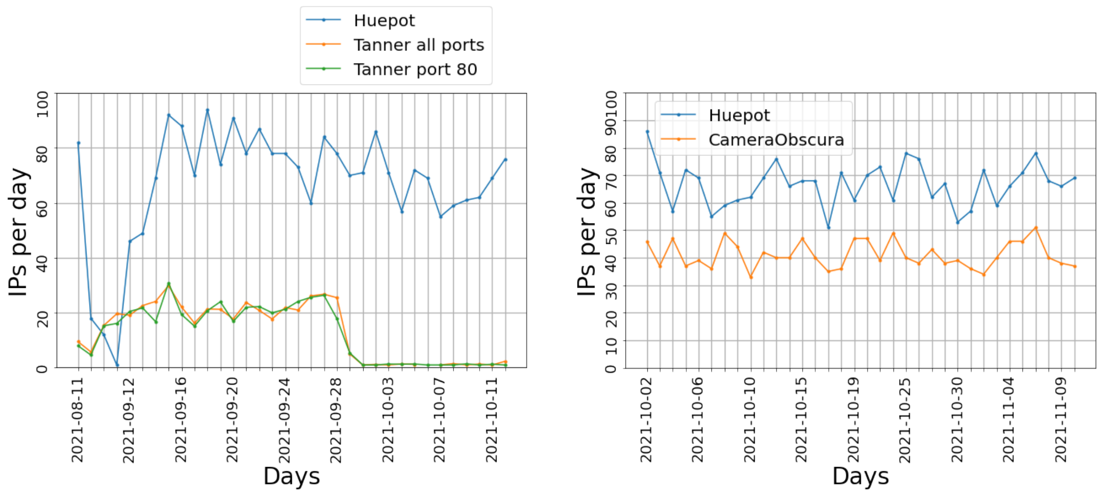
Figure 5.1: Scanners activity against HuePot

CVE_ID	HuePot	Percentage HuePot	CamObscura	Percentage CameraObscura	Tanner	Percentage Tanner	Device Type
CVE-2021-20090	11	0.065	0	0.0	0	0.0	Home Router
/boaform/admin/formLogin	634	3.754	204	4.327	77	2.829	Home Router
CVE-2018-10561	85	0.503	11	0.233	0	0.0	Home Router
CVE-2020-10987	4	0.024	0	0.0	0	0.0	Home Router
CVE-2021-29294	50	0.296	27	0.573	0	0.0	Home Router
CVE-2018-20057	4	0.024	0	0.0	0	0.0	Home Router
/setup.cgi?next_file=netgear.cfg &todo=syscmd&cmd=	98	0.58	14	0.297	0	0.0	Home Router
CVE-2021-33544	3	0.018	0	0.0	0	0.0	IP Camera
CVE-2020-25078	461	2.729	148	3.139	73	2.682	IP Camera
CVE-2019-8387	2	0.012	0	0.0	0	0.0	IP Camera
/cgi-bin/hi3510/getidentify.cgi	20	0.118	12	0.255	5	0.184	IP Camera
CVE-2019-12725	73	0.432	25	0.53	0	0.0	Zeroshell OS
/board.cgi?cmd=cd	45	0.266	0	0.0	0	0.0	NVR
CVE-2017-17106	1	0.006	0	0.0	0	0.0	Webcam
CVE-2016-5639	3	0.00018	0	0.0	0	0.0	Wireless Presentation System

Table 5.1: CVE found in requests received by HuePot and compared against Tanner and CameraObscura

5.3 Analysis of global activities

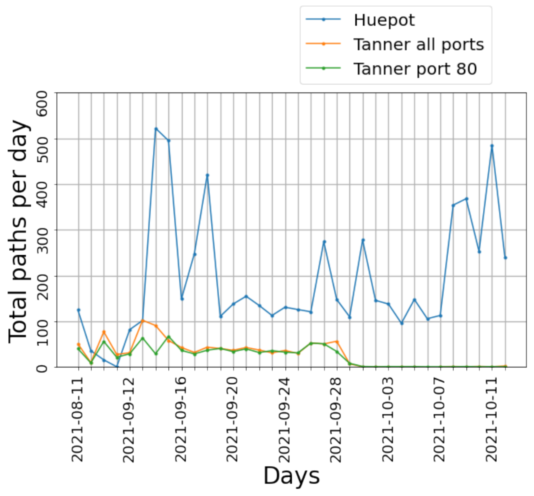
In this paragraph we study the traffic that the honeypots received per day in terms of different remote IPs, number of hits and number of visits. In the end we compute the CDF of the overall amount of paths sent from each remote host. The analysis have been performed considering the same days in which the two couples HuePot-Tanner and HuePot-CameraObscura received at least one path. Tanner has been split in two groups, one represents the set of IPs with more than one open port, a second group instead represents a set of IPs with only port 80/tcp open. The statistics related to Tanner have been averaged with respect to the number of IPs present in each group. HuePot volume of traffic observed is related to port 80 and 8080.



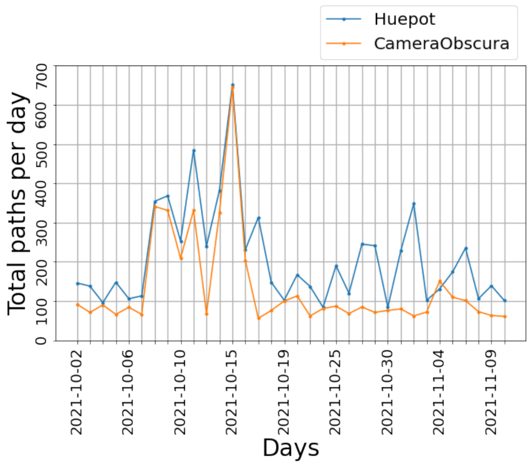
(a) HuePot vs Tanner **(b)** HuePot vs CameraObscura

Figure 5.2: Different remote hosts that contacted the honeypots per day

In figures 5.2 we report the number of different IPs that contacted the honeypots per day. In figure a we compared HuePot with respect to the two Tanner dataset. We observed that HuePot was scanned by an higher number of different IPs per day than Tanner. Both groups of Tanner after some time show a decrease in the number of different IP received, probably because the remote scanners were no more interested in its activity. In figure b we observed that HuePot number of different IPs received on port 80 daily is lower than CameraObscura, whereas globally HuePot received an higher number of different IPs.



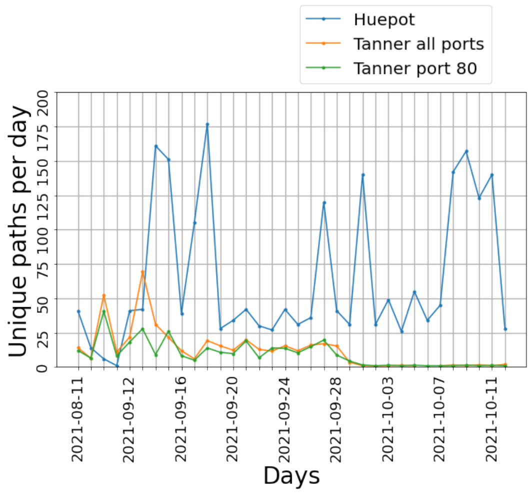
(a) HuePot vs Tanner



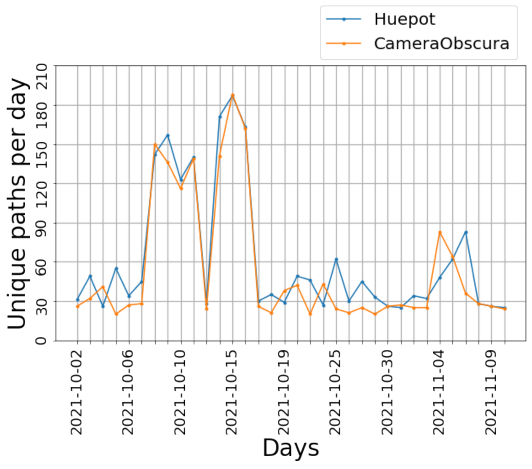
(b) HuePot vs CameraObscura

Figure 5.3: Overall number of requests received per day

In figure 5.3 we show the overall volume of traffic generated by remote hosts per day. In figure a we observe a considerable difference between the volume of traffic hitting HuePot and the two groups of Tanner. In figure b instead we observe that HuePot a lowera amount of requests on port 80 with respect to CameraObscura and the total amount of traffic hitting HuePot is higher than CameraObscura for some periods. We conclude that the similar shape in figure b could be due to the share of a group of remote scanners in common between HuePot and CameraObscura. Some of the peaks in both figures are due to the periodic scanning and bruteforce activity performed by some remote hosts that hit both HuePot and CameraObscura whereas they did not target Tanner in the period of observation. All the requests performed ended with **index.php?lang=en**.



(a) HuePot vs Tanner

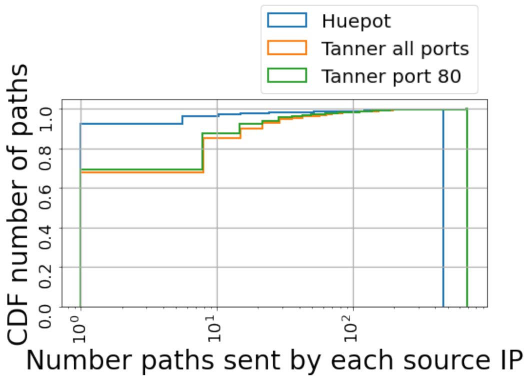


(b) HuePot vs CameraObscura

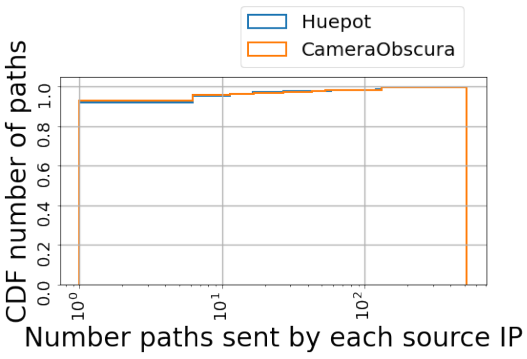
Figure 5.4: Number of unique requests received per day

In figure 5.4 we compare the amount of unique requests received per day by the three honeypots. In figure a we observe that HuePot received an higher variety of different paths per day than Tanner. In figure b instead we note that HuePot assumes values close to CameraObscura although it has two open ports. Like in the previous analysis, the peaks occurring in some days are due to the activity of scanners that performed many different brute force requests related to SQL and PHP vulnerabilities, all ending with **index.php?lang=en**.

In figure 5.5 we show the cumulative distribution function of the overall amount of requests sent from each remote host. In figure a, we note that the 90% of IPs that contacted HuePot generated a very small number of requests. On the other hand the number of IPs that contacted Tanner generating a few requests is lower than HuePot. This difference could be due to Tanner exposing more complex web pages on the Internet. As a consequence of this, when a remote scanner requested a page, it indirectly downloaded also the set of paths that compose that page. In figure b we note that the IPs produce the same amount of requests between HuePot and CameraObscura since the CDFs are overlapping.



(a) HuePot vs Tanner



(b) HuePot vs CameraObscura

Figure 5.5: CDF of the number of paths received from each remote host

5.4 Analysis of the remote hosts in common

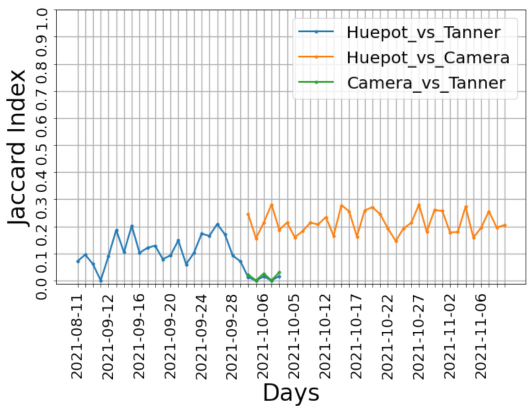


Figure 5.6: Jaccard Similarity Index of the remote hosts that contacted the honeypots per day

After an initial analysis of the global activity we investigate the amount of remote hosts in common between the three honeypots. To perform this computation we considered only the days in which a couple of honeypots received at least one path and we computed the similarity index between the sets using the Jaccard Index as follows $J(HuePot\_Tanner) = \frac{|HuePot\_rh \cap Tanner\_rh|}{|HuePot\_rh \cup Tanner\_rh|}$. From figure 5.6 it is possible to observe how in general the Jaccard Index assumes low values due to high levels of random traffic that was logged during the days. Moreover the similarity Index between CameraObscura and Tanner is very low because during the days in common Tanner received very low amount of traffic. Considering instead the pairs HuePot vs Tanner and HuePot vs CameraObscura, it comes out that the similarity index in this second comparison is higher than the first one, such that HuePot and CameraObscura had more remote scanners in common than HuePot and Tanner. This behaviour could be explained with two different hypothesis: HuePot and CameraObscura shared an higher number of remote scanners because they had consecutive IPs. As a consequence of this, when a remote host scanned one of them, also scanned the other, hence performing a side scanning. This behaviour has been shown by Soro et al. in Enlightening the Darknets: Augmenting DarknetVisibility with Active Probes (under review). Another explanation could be that they may have been seen by remote scanners as IoT devices, hence they shared some remote hosts.

Moreover we investigate the percentage of traffic generated by IPs in common between the couple of honeypots on a daily basis. The results are shown in figure 5.7, 5.8, 5.9. In both figures 5.7 and 5.8 we observe that the percentage of traffic generated by the percentage of IPs in common for each day is always lower in HuePot than Tanner and CameraObscura. This is in line with figure 5.2, where the amount of remote hosts found in HuePot during the days is always higher than the amount of remote hosts found in the other two honeypots.

In the end in 5.7 and 5.10 Tanner reaches some peaks at 100% because in those days the hosts in common generated the overall amount of requests received by Tanner. In the end 5.10 summarizes the previously obtained figures.

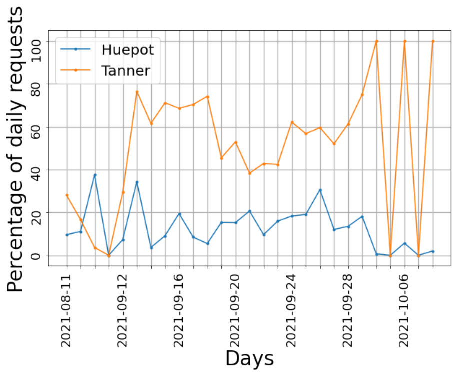


Figure 5.7: Percentage of traffic generated by IPs in common HuePot vs Tanner per day

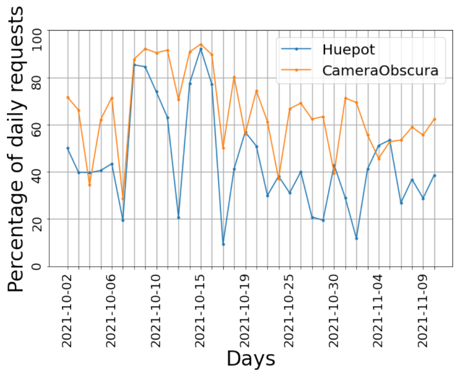


Figure 5.8: Percentage of traffic generated by IPs in common HuePot vs CameraObscura per day

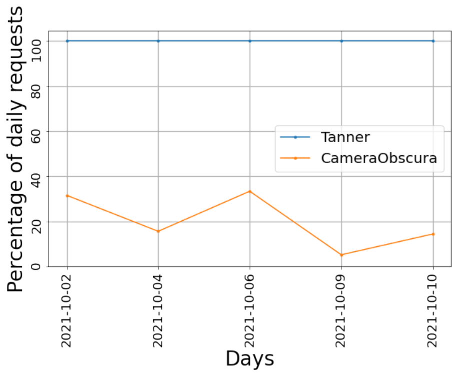


Figure 5.9: Percentage of traffic generated by IPs in common Tanner vs CameraObscura per day

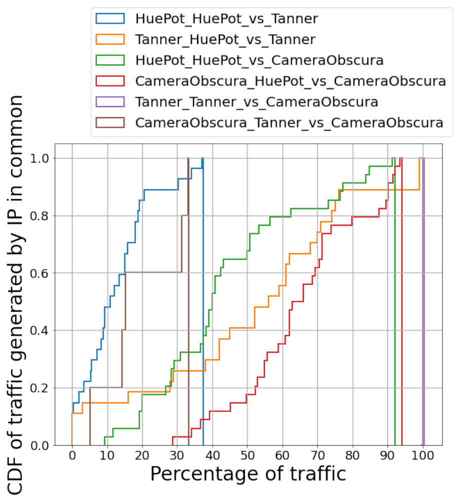
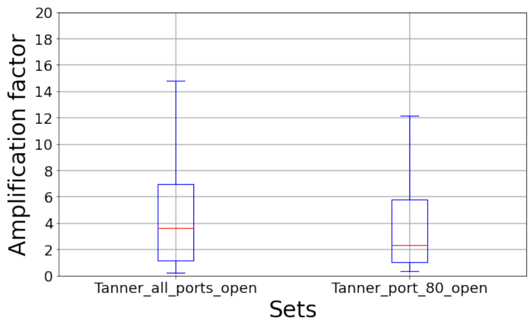


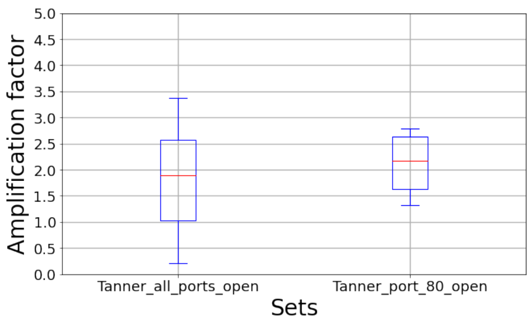
Figure 5.10: CDF of the percentages of traffic generated by IPs in common

5.5 Path amplification factor computation

In this paragraph we described more in detail the paths that HuePot, CameraObscura and Tanner received, using Tanner as baseline to make comparisons. We selected the paths in common received in the same period of time by HuePot and Tanner or CameraObscura and Tanner respectively, and we computed the ratio



(a) HuePot vs Tanner



(b) CameraObscura vs Tanner

Figure 5.11: Path Amplification Factor between HuePot and CameraObscura vs Tanner (baseline)

between the number of times a path was received by HuePot (or CameraObscura) divided by the number of times in which the same path was received in Tanner. Tanner on the other hand was divided in two groups, one group accounting for the 8 IPs with all the ports open and another group accounting for the other 8 IPs with only port 80/tcp open. We computed two different amplification factor for each comparison, averaging the Tanner groups with respect to the number of IPs present in each group to have a fair one to one comparison between HuePot Tanner couple and CameraObscura Tanner couple. The results have been shown in figure 5.11, where the red line represents the median value of the amplification factor computed for each of the four scenarios. In all the cases the median value was greater than one, such that there is an amplification factor between the number of times a path has been received in HuePot (or CameraObscura) and Tanner. On the other hand CameraObscura and Tanner had a smaller number of paths in common that were received mainly by CameraObscura and as a consequence of this, the distribution is narrower.

5.5.1 Analysis of the variation trend of the paths

In this section we investigate the trend of variation of the paths received by HuePot and CameraObscura being both emulations of IoT devices. The percentage of variation has been computed considering for every day of observation of that path the number of times it was observed in a given day divided by the number of times that path was observed the first day. We consider the maximum value of variation among all the days because we would detect how much a path could grow or decrease before it changes its variation trend. On the other hand, percentages below the 100% values have a decrease in the frequency, being the numerator lower than the denominator. In figure 5.12 is reported the CDF of the percentage of variation for all the requests (both common and not in common) received by HuePot and CameraObscura in the same time period. We observe that both honeypots have a trend of growth that on average is lower or equal to 200%. Tanner percentage of variation on average is equal to the baseline, which means that the paths did not increase day by day.

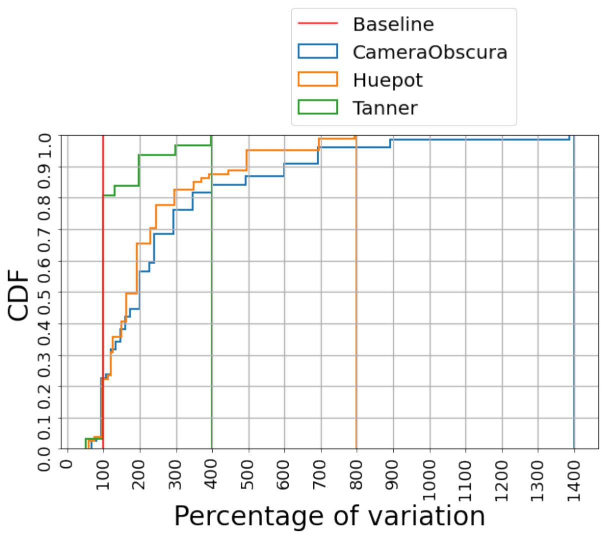


Figure 5.12: CDF of the path global variation between HuePot and CameraObscura

After the previous initial analysis we select the top 30 most received paths observed in HuePot and CameraObscura in order to focus on the tail of the distribution and compare the trend of growth with respect to the initial frequency. The results are reported in figure 5.13 and 5.14. The blue lines represents the percentage of variation of a path, whereas the orange lines represents the frequency of the first day it was observed. The red lines represent the 100% baseline: a percentage below that threshold means a decrease in the trend. In this way we showed the starting point and the trend of variation with respect to the starting point. The dots on both lines represent those paths that have been observed only in HuePot or CameraObscura. From those plots we observe that none of the paths with the highest growth are APIs of a Philips Hue or a D-Link camera. In both plots is present /config/getuser?index=0 which can be used to exploit that type of camera, but the growth are very similar in both honeypots we cannot conclude that it was sent against CameraObscura with the goal of exploit it. Moreover all the paths in the two honeypots start from a low initial frequency except for the path '/'. Indeed that path starts in HuePot with an initial frequency greater than CameraObscura and its percentage of growth is also much higher than the camera honeypot (between 350% and 400% in HuePot against around 100% in CameraObscura).

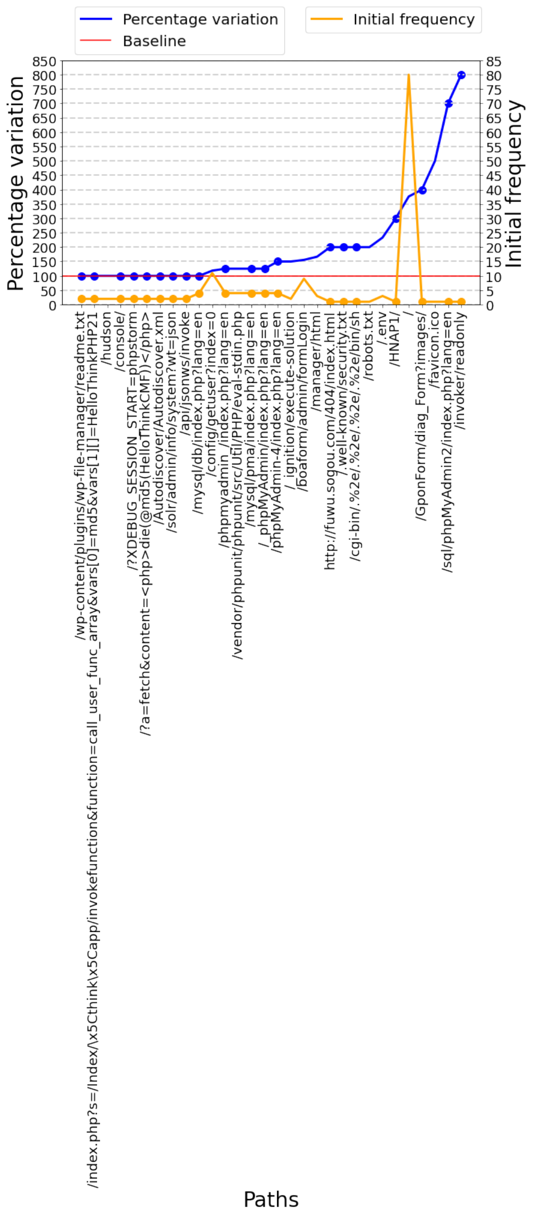


Figure 5.13: Path percentage variation of the paths received by HuePot

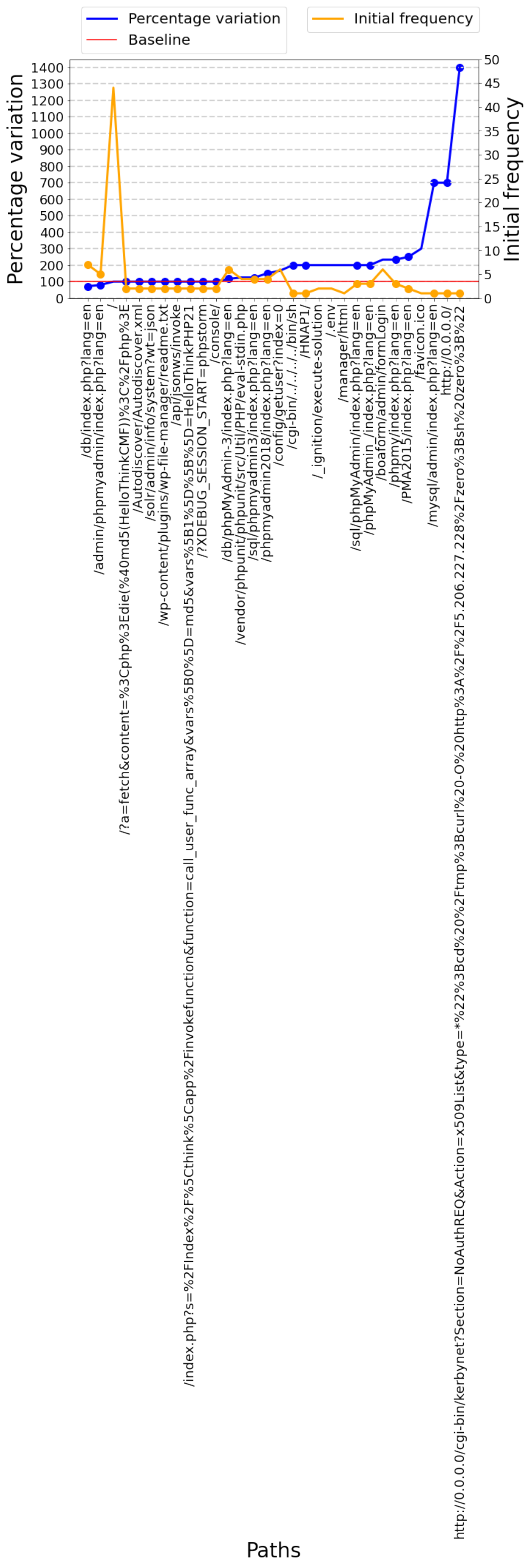


Figure 5.14: Path percentage variation of the paths received by CameraObscura

5.6 Outcomes

From the obtained results we observe that HuePot, considering traffic on both ports, collected more traffic than Tanner and CameraObscura in terms of number of requests received and number of different IPs observed. HuePot and CameraObscura requests in common with Tanner are amplified by a quantity greater than 1 with respect to the baseline Tanner and the similarity index between the remote hosts that contacted the honeypots is greater between HuePot and CameraObscura than HuePot and Tanner or CameraObscura and Tanner. On the other hand the percentage of traffic generated by the percentage of IPs in common for every day is lower in HuePot in both comparisons, meaning that HuePot received a greater fraction of traffic that was not observed by the other two honeypots, which may be due to the traffic on port 8080. In the end from the analysis of the vulnerabilities of IoT devices we observe that some of them were not found in CameraObscura or Tanner. This is the answer to the second question of paragraph 1.1.
From the analysis percentage of growth of the paths we observe that CameraObscura and HuePot had a trend of growth greater than Tanner. We can speculate that HuePot and CameraObscura have a bigger trend of growth because they send back responses that attract other traffic. A hint to this behaviour is given by the fact that Shodan tagged HuePot as an IoT device, whereas for CameraObscura it exposes a set of vulnerabilities that could be present on the device based on the device information Shodan extracts from the banner. On the other hand HuePot did not receive any API targeting it probably because there are no known vulnerabilities that can be exploited automatically by a scanner, hence we can assume that this type of device is not interesting from a scanner point of view, even though it was able to attract more traffic than Tanner. This is the answer to the third question of 1.1.

Chapter 6

Conclusion and future works

6.1 Conclusions

In this thesis we presented a new methodology to build a IoT honeypot based on the digital twin of a real IoT device. Firstly we presented the methodology applied to interact with the devices and extract the packets related to a set of actions performed by each of them. Most importantly we presented an automatic packet parsing program based on Scapy and MongoDB, which parsed the packets captured from a set of pcap files and stored them in a MongoDB database.
Then we tried to extend our knowledges about the devices through active scanning tools such as Nmap and Mitmproxy. From the passive scanning we observed that the vast majority of the packets were exchanged using TLS instead of HTTP. Each device contacted a small number of remote TCP and UDP ports, whereas they used many more different TCP and UDP ports to transmit data. Also the number of remote second level domain contacted is low. For what concern the

TLS version advertised during TLS ClientHello, all the devices used TLSv1.0 or TLSv1.2, whereas the version established during the TLS ServerHello was mainly TLSv1.2, sometimes TLSv1.0 and just one device used TLSv1.3.
After that we built a first digital twin honeypot based on the APIs of a Philips Hue Bridge, that we called HuePot. It was compared with other two honeypots that were Tanner which answered with generic web pages and CameraObscura, which was emulating a D-Link DCS-2530L camera. HuePot was contacted by an higher number of different IPs and received more requests with respect to Tanner and CameraObscura because it received traffic on port 80 and port 8080. On the other hand HuePot and CameraObscura shared an higher number of remote scanners in common because a remote host may have performed side scanning since their addresses are close or because some remote hosts recognized both of them as IoT devices. HuePot was also able to detect more requests related to IoT devices than Tanner and CameraObscura.
In the end we have performed an in depth analysis of the path received by HuePot CameraObscura and Tanner. It is important to underline that we do not have an accurate method based on a set of features to detect whether a remote host is interested to a given device or not and the work carried out in previous chapter must be considered as a first step toward the resolution of this problem. We analyzed the percentage of variation in the requests received by the three honeypots. We observed that on average the growth in HuePot is equal to the growth in CameraObscura, whereas Tanner percentages are on average on the baseline value. We can speculate that HuePot and CameraObscura have a bigger trend of growth because they send back responses that attract other traffic. A hint to this behaviour is given by the fact that Shodan tagged HuePot as an IoT device, whereas for CameraObscura it exposes a set of vulnerabilities that could be present on the device based on the device information Shodan extracts from the banner. None of the paths received

by HuePot or CameraObscura was specific of those devices.

6.2 Future works

In the future, we plan to use a methodology similar to the one described in [32] to extract features that could be useful to understand whether a scanner is interested to our honeypot. We plan to perform the digital twin of a more complex device that has more complex pages, exposes more ports (such as port 23/tcp and port 80/tcp) and exposes known vulnerabilities that can be exploited remotely. We can use the same methodology applied in this thesis to build the digital twin of such device having the goal in mind to expose more pages and more attack surfaces on the Internet. Some devices that can be used are routers and IP cameras. We can define the following categories of remote scanners:

- **Revisitor**: those scanners that send always the same requests over time.
- **Curious**: those scanners that sends different requests considering both ports over time in order to increase the knowledge of the system they are scanning without exploiting the honeypot.
- **Attackers**: those scanners that have found a vulnerability exposed on one of those two ports and actively try to exploit it.

We can extract the interarrival time between the first and the last scan performed by each scanner that was previously included in one of the three aforementioned categories. We can try to speculate whether there are differences in the interarrival time of a revisitor a curious or an attacker and whether a remote host that scanned a given port also tries to scan the other port.

Bibliography

[1] P. Grossetete. «IoT and the Network: What is the future?» In: (June 2020). URL: https://blogs.cisco.com/networking/iot-and-the-network-what-is-the-future.

[2] G. Blaine. «SonicWall: Encrypted Attacks, IoT Malware Surge as Global Malware Volume Dips». In: (Oct. 2019). URL: https://blog.sonicwall.com/en-us/2019/10/sonicwall-encrypted-attacks-iot-malware-surge-as-global-malware-volume-dips/.

[3] «IoT under fire: Kaspersky detects more than 100 million attacks on smart devices in H1 2019». In: (Oct. 2019). URL: https://www.kaspersky.com/about/press-releases/2019_iot-under-fire-kaspersky-detects-more-than-100-million-attacks-on-smart-devices-in-h1-2019.

[4] C. Cyrus. «IoT Cyberattacks Escalate in 2021, According to Kaspersky». In: (Sept. 2021). URL: https://www.iotworldtoday.com/2021/09/17/iot-cyberattacks-escalate-in-2021-according-to-kaspersky/.

[5] Antonakakis April Bailey et al. «Understanding the Mirai Botnet». In: (Aug. 2017). URL: https://www.usenix.org/system/files/conference/usenixsecurity17/sec17-antonakakis.pdf.

[6] B. Krebs. «KrebsOnSecurity Hit With Record DDoS». In: (2016). URL: `https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/`.

[7] B. Krebs. «New Mirai Worm Knocks 900K Germans Offline». In: (Nov. 2016). URL: `https://krebsonsecurity.com/2016/11/new-mirai-worm-knocks-900k-germans-offline/`.

[8] B. Krebs. «Who is Anna-Senpai, the Mirai Worm Author?» In: (Jan. 2017). URL: `https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/`.

[9] J. Franco A. Aris B. Canberk A. Selcuk Uluagac. «A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems». In: (Aug. 2021), pp. 3–5. URL: `https://arxiv.org/pdf/2108.02287.pdf`.

[10] N. Provos. «Honeyd». In: (Dec. 2007). URL: `https://github.com/DataSoft/Honeyd`.

[11] J. Guarnizo A. Tambe S. Bhunia M. Ochoa N. Tippenhauer A. Shabtail and Y. Elovici. «Siphon: Towards scalable high-interaction physical honeypots». In: (Jan. 2017). URL: `https://arxiv.org/pdf/1701.02446.pdf`.

[12] Wang Santillan Kuipers. «ThingPot: an interactive Internet-of-Things honeypot». In: (July 2018). URL: `https://arxiv.org/pdf/1807.04114.pdf`.

[13] Tongbo Luo Zhaoyan Xu Xing Jin Yanhui Jia Xin Ouyang. «IoTCandyJar: Towards an Intelligent-Interaction Honeypot for IoT Devices». In: (July 2017). URL: `https://www.blackhat.com/docs/us-17/thursday/us-17-Luo-Iotcandyjar-Towards-An-Intelligent-Interaction-Honeypot-For-IoT-Devices-wp.pdf`.

[14] Muhammad A. Hakim Hidayet Aksu A. Selcuk Uluagac Kemal Akkaya. «U-PoT: A Honeypot Framework for UPnP-BasedIoT Devices». In: (Dec. 2018). URL: `https://arxiv.org/pdf/1812.05558.pdf`.

[15] Jingjing Ren et al. «Information Exposure for Consumer IoT Devices: A Multidimensional, Network-Informed Measurement Approach». In: *Proc. of the Internet Measurement Conference (IMC)*. 2019.

[16] Arunan Sivanathan, Hassan Habibi Gharakheili, and Vijay Sivaraman. «Can We Classify an IoT Device using TCP Port Scan?» In: *2018 IEEE International Conference on Information and Automation for Sustainability (ICIAfS)*. 2018, pp. 1–4. DOI: `10.1109/ICIAFS.2018.8913346`.

[17] Mustafizur R. Shahid et al. «IoT Devices Recognition Through Network Traffic Analysis». In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 5187–5192. DOI: `10.1109/BigData.2018.8622243`.

[18] Marco de Vivo et al. «A Review of Port Scanning Techniques». In: *SIGCOMM Comput. Commun. Rev.* 29.2 (Apr. 1999), pp. 41–48. ISSN: 0146-4833. DOI: `10.1145/505733.505737`. URL: `https://doi.org/10.1145/505733.505737`.

[19] Davino Mauro Junior et al. «A Study of Vulnerability Analysis of Popular Smart Devices Through Their Companion Apps». In: *2019 IEEE Security and Privacy Workshops (SPW)*. 2019, pp. 181–186. DOI: `10.1109/SPW.2019.00042`.

[20] Davino Mauro Junior et al. *Beware of the App! On the Vulnerability Surface of Smart Devices through their Companion Apps*. 2019. arXiv: `1901.10062 [cs.CR]`.

[21] Cornelia Győrödi et al. «A comparative study: MongoDB vs. MySQL». In: *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*. 2015, pp. 1–6. DOI: `10.1109/EMES.2015.7158433`.

[22] Alexandru Boicea, Florin Radulescu, and Laura Ioana Agapin. «MongoDB vs Oracle – Database Comparison». In: *2012 Third International Conference on Emerging Intelligent Data and Web Technologies*. 2012, pp. 330–335. DOI: `10.1109/EIDWT.2012.32`.

[23] Alan Tamer Vasques and João J. C. Gondim. «Amplified Reflection DDoS Attacks over IoT Mirrors: A Saturation Analysis». In: *2019 Workshop on Communication Networks and Power Systems (WCNPS)*. 2019, pp. 1–6. DOI: `10.1109/WCNPS.2019.8896290`.

[24] Aldo Cortesi et al. *mitmproxy: A free and open source interactive HTTPS proxy*. [Version 7.0]. 2010–. URL: `https://mitmproxy.org/`.

[25] Kaizheng Liu et al. *On Manually Reverse Engineering Communication Protocols of Linux Based IoT Systems*. 2020. arXiv: `2007.11981 [cs.CR]`.

[26] *UPnP Device Architecture 2.0*. `https://openconnectivity.org/upnp-specs/UPnP-arch-DeviceArchitecture-v2.0-20200417.pdf`. Open Connectivity Foundation.

[27] Md Mainuddin, Zhenhai Duan, and Yingfei Dong. «Network Traffic Characteristics of IoT Devices in Smart Homes». In: *2021 International Conference on Computer Communications and Networks (ICCCN)* (July 2021). DOI: `10.1109/icccn52240.2021.9522168`. URL: `http://dx.doi.org/10.1109/ICCCN52240.2021.9522168`.

[28] Arunan Sivanathan et al. «Classifying IoT Devices in Smart Environments Using Network Traffic Characteristics». In: *IEEE Transactions on Mobile Computing* 18.8 (2019), pp. 1745–1759. DOI: `10.1109/TMC.2018.2866249`.

[29] *Deprecating Secure Sockets Layer Version 3.0*. `https://datatracker.ietf.org/doc/html/rfc7568`. Internet Engineering Task Force.

[30] *Deprecating TLS 1.0 and TLS 1.1.* `https://datatracker.ietf.org/doc/rfc8996/`. Internet Engineering Task Force.

[31] Paracha, Muhammad Talha, Dubois, Daniel J, Vallina-Rodriguez, Narseo, Choffnes, David. «IoTLS: Understanding TLS Usage in Consumer IoT Devices». In: (Nov. 2021).

[32] Johan Mazel, Romain Fontugne, and Kensuke Fukuda. «Profiling internet scanners: Spatiotemporal structures and measurement ethics». In: *2017 Network Traffic Measurement and Analysis Conference (TMA)*. 2017, pp. 1–9. DOI: `10.23919/TMA.2017.8002909`.