University of Kent

A systematic analysis of attacks on a honeypot

CO880 - Project and Dissertation

Yann Ferrere
ID 15906875
yf57@kent.ac.uk

Approximately 9,000 words

Abstract

Attacks against servers and in particular Linux servers are becoming increasingly common. However, their protection is necessary in order to maintain the normal functioning of all services that all users, who have a device connected, use. Indeed, websites, mobile apps emails, and a lot of other services need servers in order to work. Moreover, it is important to notice that attackers are usually one step ahead of security researchers. But some techniques are used to understand attackers behaviours in order to develop new ways to protect systems against them. A honeypot is one of those techniques. It consists of emulating vulnerable servers designed to collect attacks and analysed them in real time. Consequently, this paper will describe the usage of a Cowrie SSH honeypot, that has been put in place and run during 6 months in order to collect attacks information and samples of malicious files. Moreover, this paper will also present a pre-analysis static technique to examine malware. This analysis method has been performed by using hard-coded strings from gathered malware sample.

Acknowledgement

I would like to thank my supervisor, Dr Andy King, for his guidance during all stages of my researcher and for its knowledge sharing in malware analysis. Thank you also to Thomas Seed for advising me with its expertise in machine learning. Moreover, I would like to thank VirusTotal team to gave me a free access to their tool in order to perform my malware analysis. And finally to Jeremy Pouyet for providing me information about neuronal networks application.

Table of contents

1 Introduction **5**
- 1.1 Context . . . 5
- 1.2 Goals of the project . . . 5
- 1.3 Contribution . . . 5
- 1.4 Structure of the disseration . . . 6

2 Literature review and related work **6**
- 2.1 Honeypots . . . 6
  - 2.1.1 Overview . . . 6
  - 2.1.2 Case of studies . . . 7
- 2.2 Malware . . . 7
  - 2.2.1 Overview . . . 7
  - 2.2.2 Case of study . . . 8
  - 2.2.3 Analysis methods . . . 8

3 Implementation and configuration **9**
- 3.1 Honeypots . . . 9
  - 3.1.1 Honeypots overview . . . 9
  - 3.1.2 Cowrie and its configuration . . . 10
  - 3.1.3 Kippograph and data vizualisation . . . 11
- 3.2 Malware string analyzer . . . 12
  - 3.2.1 Strings categories . . . 12
  - 3.2.2 Strings extraction . . . 13
  - 3.2.3 Virus total analysis . . . 14

4 Honeypot results **15**
- 4.1 Attacks overview . . . 15
- 4.2 Authentication attempts . . . 17
- 4.3 Malicious downloads . . . 18
- 4.4 Malware binaries analysis . . . 19

5 Future research **20**

6 Concluding discussion **20**

7 Bibliography **22**

8 Appendices **24**

List of Figures

1 Human activity per week - Kippograph . . . 11
2 Number of connections per unique IP address and country codes - Kippograph . . . 12
3 Ouput string processing - Malware Strings Analyzer . . . 13
4 Number of connections per unique IP (TOP 10) - Kippograph . . . 15
5 Number of connections per country - Kippograph . . . 16
6 Top 10 input (overall) - Kippograph . . . 17
7 Overall success ratio - Kippograph . . . 17
8 Top 10 username-password combinations - Kippograph . . . 18
9 Positive malicious file output - Malware Strings Analyzer . . . 19
10 Positive malicious IP address output - Malware Strings Analyzer . . . 19
11 Overall honeypot activity - Kippograph . . . 24
12 Overall post-compromise activity - Kippograph . . . 24
13 Top 10 passwords - Kippograph . . . 24
14 Top 10 usernames - Kippograph . . . 25
15 Most successful logins per day - Kippograph . . . 25
16 Successes per week - Kippograph . . . 26
17 Successful logins from same IP (TOP 20) - Kippograph . . . 26
18 Top 10 SSH clients - Kippograph . . . 27
19 Top 10 successful input - Kippograph . . . 27
20 Top 10 failed input - Kippograph . . . 28
21 Number of connections per unique IP (Top 10) - Kippograph . . . 28

1 Introduction

1.1 Context

With the development of the Internet, the number of computer servers has significantly increased. Indeed, Internet is based on interconnection between all devices such as computers and smartphone but also servers. The aim of those servers is to store data such as video, image, music, databases, but it also provides interaction between software executed on them and remotes users. These interactions are called client/server communication. Servers can, for instance, execute a software such as a web server, which consists of making accessible web pages stored on it to remote users. Evidently, plenty of other servers usages exists but all of them widely implies to store data, run software and communicate information on the network.

Meanwhile, the usage of devices connected to internet has become extremely widespread. Some basic tasks of daily living, such as consulting its bank accounts, communicate with each other or read the news, can be performed anywhere by using an internet-connected device, such as a smartphone. All of these services can be provided by a private company, but also by a government or an individual. In order to function efficiently, these services are developed by using a coding language and are hosted on a server, to store users data and to be accessible remotely.

However, all of these communications and stored data can be more or fewer sensitives. Indeed, communication between two people (bank account information, password, and other credentials data) have a value that is interesting to steal or alter by an ill-intentioned person.

1.2 Goals of the project

The risk of a server attack is now becoming ubiquitous. Based on this known risk, I decided to analyse what are the steps of an attack against a server. First, it is important to notice that servers work as common computers and need to have an operating system (OS) to operate properly. This dissertation will focus on Linux servers distributions. Indeed, the majority of servers are Linux based and can run, for instance, known OS such as Ubuntu [1] or Debian [2].

Consequently, this project consists of putting in place a Linux server accessible remotely by its unique public IP address and then to install a Honeypot on it. This type of tool is a software that emulates a fake Linux environment in order to analyse attackers behaviour. Moreover, a honeypot makes a server accessible remotely through a fake SSH connection with really common identifiers (username/password) in order to be brute forced easily. Hence, this tool provides a way to allow malicious connection on its server while restricting the access to the machine.

During 6 months, our honeypot server received around 82000 SSH login attempts with 62 percent of successful connection. Those attacks were from 2700 distinct IP addresses and from a large number of different countries, all around the world. Following these attacks, multiple data such as command lines executed by attackers, scripts and malware downloaded, have allowed me to perform a more detailed analysis of attackers behaviours and their motivations.

1.3 Contribution

The honeypot technique that consists to collect attacks on a Linux server, is not new. Several papers have already put in place this configuration in order to analyse attackers behaviours. However, there are plenty of ways to implement a honeypot. Some papers have created their own network architectures with multiple machines [3], while others have used existing solutions such as Kippo [4]. In this dissertation, the Honeypot is cowrie [5], a fork of the kippo project, which is not longer maintained. Consequently, this paper shows, in a real case scenario, that this type of honeypot can be an appropriate choice for a project that needs to collect data from Linux server attacks.

It is also important to note that attackers technics to compromise Linux server are constantly evolving. Indeed, security researchers and developers are continuing to look at what are the newest vulnerabilities and try to patch them (CVE [6]). Then, attackers find new attacks scenario in order to bypass new security detection and protection tools, and so on. However, security researchers have to put sensors to collect data and analyse security issues.

Consequently, all along this project research, a database containing all attacks information has been created in order to update and improve knowledge of current Linux server attacks.

Furthermore, because of a large number of ELF malicious collected binaries, I performed a static analysis of them by analysing strings. In fact, when malware are not packed or stripped, multiple hard coded strings can be extracted from them. During the analysis of these strings, 10 differents types were observed such as IP addresses, URLs or readable English messages. In order to automate their extraction, I developed a python script that automatically extract strings from a given malware, sort and store them into a database. The aim of this could be to generate a large dataset from strings contained in malware to be applied through a data mining algorithm, in order to perform a pre-analysis of a malware.

1.4 Structure of the disseration

In order to understand the functioning of Honeypots and their objectives, a short review of the existing literature about similar already performed project, is exposed first.

Then, the technical implementation and configuration of the server, the honeypot and tools used to store, and read collected data will be examined in details. In the same way, the development and usage of the malware strings analyser will be shown.

A concluding discussion will summarise this analysis and results will be described in the same section. Finally, in a future research part will provide a way to a possible future development and reflexion.

2 Literature review and related work

2.1 Honeypots

In this section, we will review articles concerning Honeypots, a technique allowing security researcher to analyses attacks on Linux server. First of all, we will see articles that achieve an overview of honeypot technology. Then, we will examine articles performing case of studies where authors analyze attacks through a honeypot system.

2.1.1 Overview

This sub-section present a review of an article called "The Honeynet Project: Trapping the Hackers" [7] presenting an overview of honeypot systems installed on Linux machines.

This paper presents the honeynet project, a security research organization composed of international security professionals. The purpose of this organization is to analyze the black-hat communitys attacks by examining their tools, tactics and more generally their behaviors. Based on this willingness to collect data from attacks and learn from them they decided to create a network, also called honeynets, in order to perform this task. A honeynet is basically a type of honeypot which consists of putting in place a system (without any production value and that doesnt contain any sensitive data) designed to be attacked and/or compromised. They can be used by an organization to improve its security (production honeypots) or by the researcher for gathering information on attackers (research honeypots). However, authors of this article underline that in all cases of usage, a honeypot cannot replace security technologies because of the risk to contain vulnerabilities that can be exploited. Finally, honeynets are research honeypot that doesn't emulate any services, but it implements real systems with hidden software allowing to manage and capture data from attacks. Consequently, the risk of this type of honeypot is that attacks have to be limited to the tested environment and dont impact to any other systems (data control). But it also consists to ensure that data from attacks can be detected and captured even if they are obfuscated or encrypted (data capture). An improvement for this paper could have been to perform a case of study of a honeynet in a real case scenario and detail more precisely what data can be captured and how to interpret them.

2.1.2 Case of studies

This sub-section present review of two articles that put in place Honeypots system in order to collect data from remote attacks on Linux machines.

Attackers profiles and behaviors In this part, we will see an article called "Analyzing the process of installing rogue software" [8] with an analysis of their results of their Honeypots gathered data, focused on attackers behavior and profiles.

This paper has been written in order to report results of an experiment performed to understand and analyze malicious behavior following a remote Linux server attack. In order to realize this analysis, authors of this article have implemented four honeypots. Those honeypots are four Linux target computer using SSH with a simple password, easy to find for an attacker. Moreover, that machine also contains monitoring software such as a modified OpenSSH server, syslog-ng, and strace to gather a maximum of data during a potential malicious access to the computer. By using this system authors have collected a large amount of data and choose to divide them by using two concepts. The first one is called session and define all distinct SSH interaction between an attacker and one of their honeypot. The other one called action corresponds to a sequence of command line execute by an attacker to do a specific task such as gathering information or install malicious programs. Furthermore, those honeypots were configured to record all network traffic and consequently, authors were able to collect malware downloaded by attackers on compromised computers. Consequently, they decide to classify them in categories corresponding to their types such as IRCBots, Rootkit, and Flooder. This classification has been realized at 50% by using VirtusTotal [9] and the 50% remaining percent by a manual identification using their source code. However, this paper doesnt provide more details about the manual malware reversing.

Analysis of collected malware In this part, we will review an article called "Characterizing Attackers and Attacks: An Empirical Study" [10] with an analysis of their results of their Honeypots gathered data that also performed an analysis of downloaded malwares.

This paper consists of an analysis of collected data from SSH honeypots configured with weak usernames and passwords, intended to be found by attackers using brute force attacks. Authors of this article decided to use an SSH honeypot called Kojoney SSH honeypot [11] in its version 0.0.4.2 and all data accumulated during this experiment were stored in a central database. By using data stored in this database, they were able to perform an analysis of two main aspects of a Linux server attack. First of all, authors of this articles performed a general overview of attackers behavior by analyzing gathered data from their honeypots. These parts consists of determining where come from attacks (by using geographic location of IP addresses), what are the most common username/password combination, what are the OS used by the attackers and what is the global activity of attempted connection on the SSH ports (ratio successful and failed authentication by their source IP). The second asset aspect concern the activity inside the Honeypot after a successful connection on the SSH port. In this part, authors were able to determine favorites commands used by attackers but also collect malware downloaded by them. After a more detail analysis of that malware they authors retrieve their antivirus names, a platform which is necessary to run them and sometimes botnets information (channels, username, password,). Although, that the analysis of gathered data was detailed authors doesnt explain what are the steps used to perform all of their malware analysis.

2.2 Malware

In this section, we will review articles concerning malware, malicious software which are used by attackers to compromised machines and perform actions remotely.

2.2.1 Overview

This sub-section present a review of an article called "Introduction Linux based malware" [12], presenting an overview of malicious softwares used by attackers.

The aim of this paper is the analysis of malware on Linux operating system. Indeed, as explained by the authors, Linux systems are particularly vulnerable to malicious software because of their minimal defenses against them.

Moreover, those systems are increasingly used in particular in IoT (Internet of Things) embedded devices and servers. In this paper malware are defined as Code used to perform malicious actions. Moreover, authors underline that malware needs to exploit a vulnerability in order to compromise a machine. This article also specifies that malicious programs can embed, in many cases, several functions such as propagation mechanisms and command and control functions. In addition, authors define the notion of the payload as a code to exploit a particular vulnerability, generally present on malicious programs in order to infect the targeted system. In order to reduce the number of malware attacks, several systems can be put in place to filter traffic to and from the machine to block and/or detect malware. For instance, web application firewall (WAF) configured as an HTTP reverse proxy can protect a web server from potential attacks. In the same way, network-based monitoring programs such as SNORT are necessary to assure a minimal protection of a system. However, this article could be improved by describing malware classification technics in order to report and monitor in the best manner potential malware attacks on its Linux system.

2.2.2 Case of study

This sub-section present review of an article called "Psyb0t Malware: A step-by-step Decompilation case study" [13] that performed an analysis of a known malicious software called Psyb0t.

The authors of this paper perform an analysis of the malware PsyB0t by using a decompilation technique which consists of retrieving a readable code based on the machine code of a binary. However, usage of decompilation tools on malware is a challenging task. Indeed, malware doesnt respect applications standards in order to be complex to reverse engineer. For instance, a malware has usually stripped symbols, its code is obfuscated and can contain polymorphic code. Moreover, malware targets multiple platforms and consequently can be compiled for each of them. During the analysis of the PsyB0t malware, authors were able to unpack it by using UPX tool. Then, they process the whole executable to retrieve strings (printable characters that terminated by a zero byte
0) and some symbols by analyzing data sections of the binary. Next, authors used the address of the programs main by using its internal computer-specific database or using a heuristic detection. Based on that they were able to create a control-flow graph that examines all branch instructions to recognize conditional and unconditional branches, functions calls and returns from functions. At this steps the assembly code contains many redundant instructions, consequently, authors performed optimizations such as detecting multiple instructions that correspond to a known function and replaces them by the function name. Finally, the last step is to use the produced input of the last step to produce code in the specified language syntax (C or Python like). By analyzing the code authors were able to detect functionalities of this malware (DDoS, brute-force, downloading of files,) and much other useful information. Finally, this paper produces a complete analysis of the PsyB0t malware, however, it would have been interesting to perform an overview of decompilation tools and justify their choice to use a retargetable decompiler that is being developed within the Lissom project.

2.2.3 Analysis methods

This sub-section present review of articles that examine methods in order to perform analysis of malicious softwares by security researchers.

Static and dynamic analysis In this part, we will review an article called "Automating Linux Malware Analysis Using Limon Sandbox" [14], concerning static and dynamic reverse engineering methods allowing security researcher to analyze malwares.

This paper focuses on the presentation of Limon Sandbox, an automating Linux Malware analysis tool developed by Monnappa K A. This sandbox, written in Python, allows the user to automatically collect, analyze, and generate reports containing information of processed malware. Moreover, this software aggregate knew malware analysis tools such as YARA, VirusTotal, and the Volatility memory forensics framework. Limon sandbox performs three common types of malware analysis. First of all, a static analysis of the malware is performed without executing the malware. This step can able the user to learn multiple useful information about the malware such as the file type, its cryptographic hash, Strings embedded within the file, obfuscation methods, Then, a dynamic analysis is performed. This step consists of executing the malware sample in a safe environment (a virtual machine without any personal data) and where it is possible to monitor as it runs. By using these technics a security researcher will be

able to get more information which would otherwise be impossible to gather with a static analysis such as process, file system and network activity. Next, Limon sandbox allows users to realize a memory analysis of the previously ran virtual machine. This step allows users to extracting forensics information such as running processes, network connections, loaded modules, API Hooking, Finally, these tools generate a report in text format that contains all previously collected information. This paper provides a complete explanation of Limon Sandbox tool, however, it might be interesting to detail how can be interpreted results generate by the report.

Unpacking In this part, we will review an article called "Revealing Packed Malware" [15], concerning packing method that malicious softwares developers used to bypass antivirus detection.

The starting point of this article is that during the past few year malware threats have increased significantly. In order to protect users against this threats multiple companys developed antivirus software to detect and block malicious programs that try to compromise a computer. However, malware authors use packers in order to evade their detection by antivirus. The author defines a packer as a binary tools that instigate code obfuscation. Currently, modern malware can completely bypass personal firewalls and antivirus scanners by implementing this technic. A malware can use a packer software in order to compress and encrypt itself. Then, when the packed files are loaded into the memories it is able to restore the original executable and run its malicious part. Consequently, antivirus that tries to match the signature of the packed malware with the known virus does not detect any risk. Packers can be classified into four categories:

1. Compressor: that shrink file size with little or no anti-unpacking tricks (i.e. Upack UPX).
2. Crypter: that encrypt and obfuscate the original executable and prevent to be unpacked without any compression (i.e. Yodas crypter).
3. Protectors: that combine features from compressors and crypters (i.e. Armadillo).
4. Bundlers: that pack a software package of multiple executable and data files into a single bundled executable file (i.e. PEBundle).

In order to answer to this threat of packed malware, security researchers and antivirus editor use unpacking technics to inspect the original executable signature. One manual technics is to use debugger tools such as Ollydbg. However, automate packers detection exists and are usually used by antivirus that develops static unpackers. This type of tool consists of dedicated routines to decompress and decrypt executable packed by specific packers without executing the suspicious programs.

3 Implementation and configuration

3.1 Honeypots

As we have seen previously, the analysis of Linux servers attacks requires collecting the maximum of information about each of them. In order to perform this task, I choose to put in place a honeypot software called Cowrie [5]. In this section will be explained how works a honeypot, how cowrie has to be configured and finally how it is possible to extract useful attacks information from all collected data.

3.1.1 Honeypots overview

As seen during the literature and related works review, Honeypots are not a recent technique and are still commonly used to trap attackers. Moreover, they can be used in multiple manners to analyse different types of attacks. However, all of them are aimed at being undetectable by attackers. Indeed, honeypots software are built to receive attacks from real attackers who wants to compromise a vulnerable machine and confine the attack to a controlled. environment. Hence, the configuration part of this type of tool and its security has to be constantly monitored. A honeypot containing an unknown vulnerability (also called 0day) or with incorrect settings, could be exploited by attackers and consequently lead the machine to be compromised.

Multiple systems can be used and some honeypots project, for instance, are able to be installed on smartphones by using a tool such as HoneyDroid [16]. Moreover, the HoneyNet Project [17] gather security specialist that

use multiple types of honeypots in order to collect data and report their knowledge about detected attacks on all platforms. They also developed various and sundry tools such as HoneySnap [18], a script used to extract malicious events from large logs generated by their honeypots, and allowing to understand attacker's methodology on the inside of the compromised system.

Honeypots can also be associated together and create HoneyNets. Unlike a honeypot that can be installed on a system in production inside a complete network as a sensor or alone on a a separate machine, a honeyNet is composed of multiple honeypots. Each of them can allow to detect and analyse more complex attacks in a large network closer to reality for a company than a single machine.

In order to be attacked, a honeypot has to be vulnerable and accessible remotely. There are a large number of possible vulnerabilities depending on the system targeted. Each of them can allow attackers to execute more or less critical tasks such as obtained a root access to a server. A known honeypots technics, to perform this tasks, is to emulate a fake Secure Shell (SSH) access. This service is used to obtain a remote secure shell from a server, in order to monitor it. Indeed, this remote shell can be accessed by knowing a username and its associated password, available on the server. Once the honeypot emulates a fake SSH access it will allow creating multiple valid identifiers allowing an attacker to access to a fake Linux environment. The objective is to set a very common username and password to facilitate the access to an attacker.

3.1.2 Cowrie and its configuration

Cowrie is an SSH honeypot allowing its user to log brute force attacks and the entire shell command lines performed by attackers once logged on the server. Developed in Python by Michel Oosterhof, this project is a fork of another well known SSH honeypot called Kippo [19] and developed by desaster. Indeed, the developer of Cowrie decided to add many features such as the addition of the "ssh exec commands" support. Indeed, it is possible to execute command lines through SSH after being connected or execute a single command line remotely, without obtaining a full shell, but only the output of its command. This is commonly used by attackers that developed scripts to automate their attacks and is consequently useful to be implemented on an SSH honeypot. Moreover, because the attackers who access to cowrie is in a fake Linux environment jailed from the rest of the real server operating systems, all command lines have to be re-implemented. Consequently, Kippo and then Cowrie honeypots have implemented commons command lines used by attackers and more generally all Linux users. However, one of them called "ping", used to test the communication between two devices on a network allowed in Kippo to ping IP addresses such as "555.555.555.555" which is an impossible IP address. Indeed, an IP address is composed of 4 bytes and bytes can only store a value between 0 and 255. Hence, Kippo project is no longer maintained and some automate scripts used to detect this honeypot have been developed.

Consequently, I decided to use this SSH honeypot in order to collect attacks data. Its installation is really basic. First of all, a server accessible by a public IP address is needed. For my research, I choose to subscribe to a Virtual Private Server (VPS) on OVH [20], a French servers provider. Unlike a dedicated server, a VPS is basically a virtual machine running on a physical machine with a chosen operating systems. Its advantage is its low cost, because of the multiple virtual machines that can be run at the same time on a single dedicated server , associated with a sufficient configuration for a honeypot (2gigabytes of RAM and 10 gigabytes of storage).

The second part consists of deploying the honeypot on this VPS. Cowrie is available on Github [21], a website allowing to store developers projects on a private or public repository with versions numbers. Hence, it just needed to install with the packet manager available on the chosen operating system the "git" command line in order to clone the cowrie repository. This previous step implies to have created a user "cowrie" on your system, only use to run the honeypot. Then, we have to install python and the associated libraries listed in the requirements.txt file. Once it has been done, the configuration file has to be edited to enable the MySQL [22] database storage of all data collected by the honeypot and also set the port number of the honeypot to 22. Indeed, the SSH service is by default set in port 22, consequently, the server has to be configured to ensure that the SSH honeypot run on port 22 and the original SSH server on a chosen port. This step is essential to be sure to maintain an SSH access to monitor your server.

Before running this honeypot we can also define a list containing valid usernames and passwords to be connected to the server. Then, a shell script called "start.sh" has to be executed to run the honeypot. More details about the installation and configuration can be found easily on internet.

Finally, one important point is to ensure that the server use does not contain any sensitives information. Indeed, this is a software and despite the availability of the code on GitHub the authors cannot ensure that any vulnerabilities will be found.

3.1.3 Kippograph and data vizualisation

An SSH honeypot generate a large amount of data. These data may refer to all information related to SSH connection attempts such as identifiers used and IP address used, but also to all events when the attackers access to a shell on our server. In the case of cowrie and Kippo Honeypots, all data are stored on a MySQL database automatically. This database is structured in 8 tables (auth, clients, downloads, input, keyfingerprints, sensors, sessions and ttylog). However, due to the quantity of attacks that can be received on an SSH honeypots, it quickly became apparent that read information from a database was a difficult task. Consequently, I decided to put in place a graphical interface allowing to extract and format useful data contains on this database.

Moreover, it is important to notice that Cowrie apply the same database structure as Kippo. Hence, due to the large community of Kippo, I have been able to use Kippograph (v1.5.1) [23], a tool from their toolbox. This tool is a web interface using the Libchart PHP chart drawing library that gathers multiple libraries in order to build maps and graphs from given data. Consequently, this tool allowed me to perform a daily analysis of the actual state of attacks performed against my server. Indeed, a simple GET request to one of the 6 available pages (kippo-graph, kippo-input, kippo-playlog, kippo-ip, kippo-geo and graph gallery), return several graphs and information such as the actual Human activity per week (see Figure 1) and the number of connections per unique IP address with their country codes (see Figure 2), easy to understand and interpret.

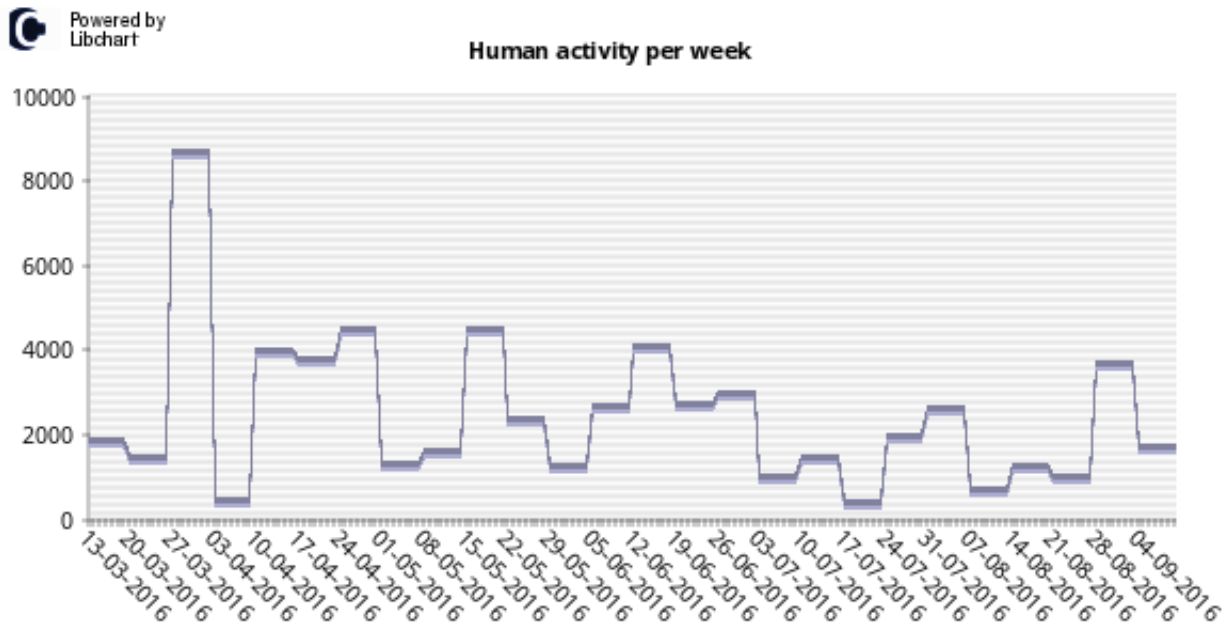

Figure 1: Human activity per week - Kippograph

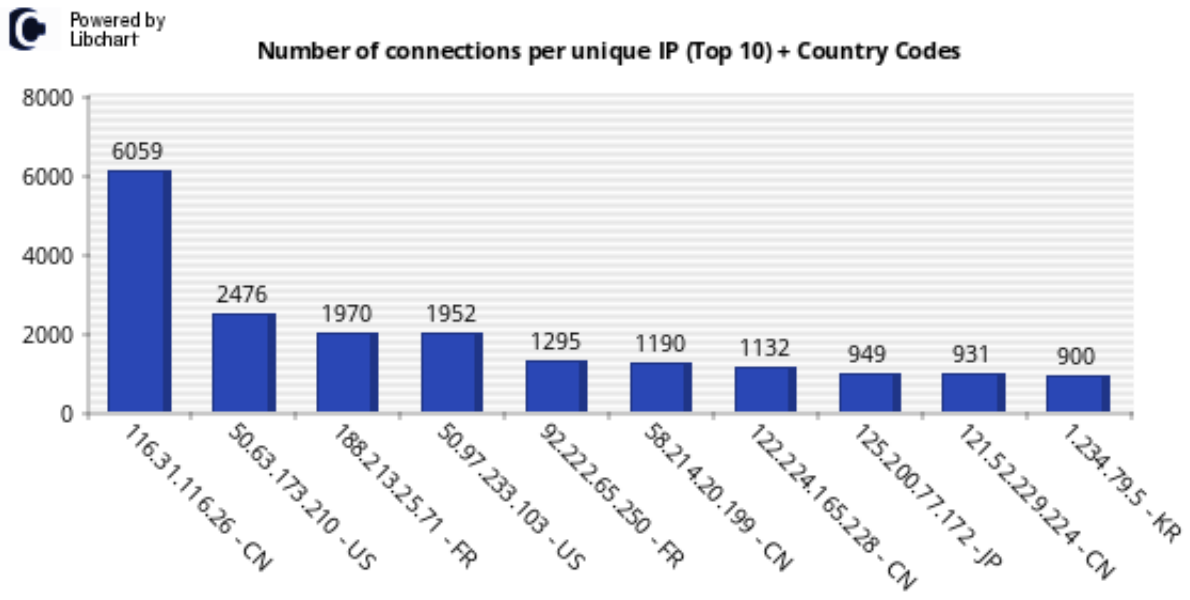

Figure 2: Number of connections per unique IP address and country codes - Kippograph

The installation is also simple and fast to implement. First of all, Kippograph is a web interface consequently a webserver such as Apache [24] or Nginx [25] has to be installed on the server. Once, it has been installed all of the source code, located on GitHub has to be downloaded on the web server localisation (by default /var/www/html). Then, the configuration file has to be edited to allow it to get data from the cowrie MySQL database. Finally, this website has to be protected with all basic security rules such as a strong password to avoid everyone to access to this information remotely. More details about the installation and configuration can be found easily on internet.

3.2 Malware string analyzer

By putting in place an SSH honeypot, I have been able to collect a large number of binary downloaded by attackers when there were connected to the server. Those executables and Linkable Format binaries, also called ELF binaries, contains hard coded strings on them. Based on these strings I decided to implement a python script, called Malware Strings Analyser (MSA) [26], used to extract and sort strings into categories. This allowed me to automate this task to all gathered binaries and consequently to focus on the analysis of data generated by this script. The source code and documentation are available on GitHub.

3.2.1 Strings categories

The analysis of readable strings is useful in order to understand how works a malware without running it, also called a static analysis. However, this could be done only if binaries are not packed. Indeed, some malware developers can employ encryption algorithms in order to hide the active part of the malware to bypass possible detections of them. Moreover, if a binary is stripped then we could not be able to get readable strings such as the name of variables and functions symbols. Indeed, ELF binaries are commonly compiled with debug information. However, developers can use compilers options in order to remove all debug information in order to hide a maximum of information about their functionalities (included libraries, functions used,...).

During my research, I divided strings that I found in 10 categories:

1. IP addresses: it is possible to find hard coded IP addresses, with sometimes a port number (e.g. 93.174.89.143:23)
2. Identifiers: some binaries contains strings which look like username/passwords, probably to perform a password guessing brute force attack on the compromised server.

3. Command lines: there are some strings that contain readable shell command line that could be executed by the malware.
4. Url and files: it is possible to find strings that contain URL allowing to download a file, probably to update the malware remotely. Those URLs are often written into a command line string.
5. Path: some binaries contains strings with the absolute path to a Linux binary or a file (e.g. /bin/sh)
6. Symbols: when a binary is not stripped it is possible to get more specific information such as the name of used functions.
7. Format string: this type of string consists of building string dynamically (e.g. %d.%d.%d.%d for an IP address).
8. Display message: some readable message contains error, success or helps message. Those string can help to understand functionalities of the malware.
9. Sections: it is possible to get name of the program sections (e.g. .text, .data, .bss, ...).
10. Other: Some other strings contain readable information but I was not able to determine if they are useful and where they come from. This category also contains some extracted strings with random character without sense.

Consequently, Malware String Analyzer tool, extract all readable character contains in a binary file (see Figure 3), sort them depending on their groups and store them in an SQLite database [27]. Finally, it is important to notice that a malware's developer can also introduce incorrect readable strings in order to hide the correct functioning of its malware.

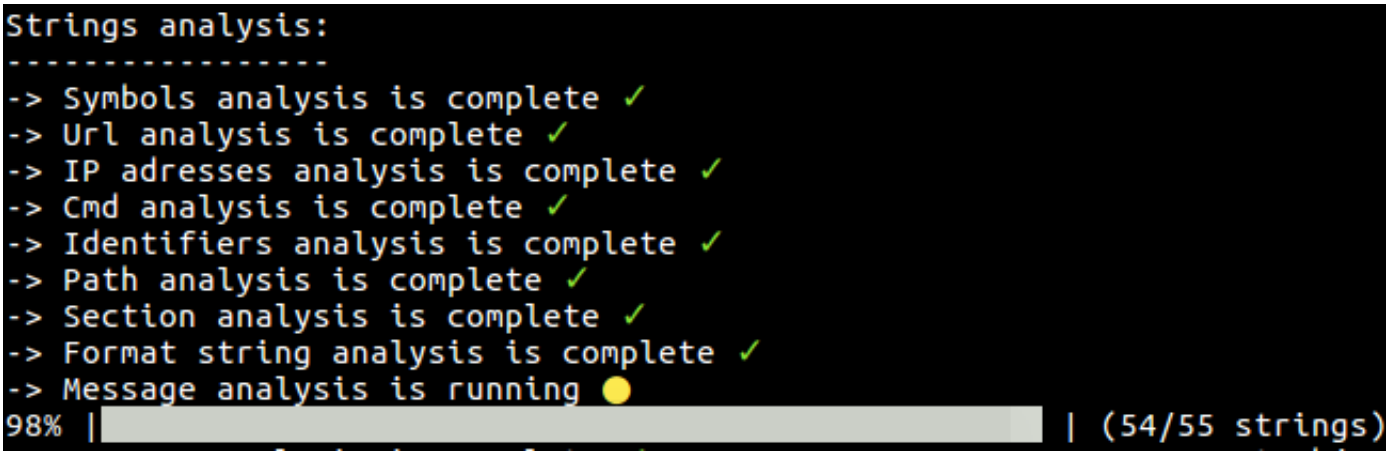

Figure 3: Ouput string processing - Malware Strings Analyzer

3.2.2 Strings extraction

Malware string analyser tool is used to extract strings in order to find patterns and link between those information and to perform a pre-analysis of a given binary file. The generated database contains a table for each binary analysis with all strings sorted by their type. Those data could be used to determine a percentage of chance that a binary file is or not a malware.

In order to extract these strings, I determine basic criteria representing a string. Indeed, a binary file is a basically a file containing bits (0 or 1). By grouping them into group of 8 bits we obtain a byte. This byte can contain values between 0 to 255, and consequently, it can represent a character from the ASCII (American Standard Code for Information Interchange) table. This table is a standard of character encoding and contains all characters necessary to write in English. Based on it I decide to extract all readable character from this list: \: , [ ] <>% $ _ and also the following: space, A to Z (in upper and lower case). Hence, the malware string analyser script starts by opening a file and search for strings from the previously listed characters with a size of minimum 4 characters, one after

the other. The method to extract these data is called a regular expression which is a specific syntax that allow to match it with multiple corresponding strings.

This first step enables the script to collect all readable strings. Afterwards, a specific python function is applied for each string categories. In order to code these sorting functions, I used the following 4 technics:

1. Regular expression: as during the extraction of strings from a binary, some string are defined as belonging to categories by using a specific regular expression. Indeed, the categories IP addresses, URLs, Path, Format string and Sections are extracted from raw strings by formatting a specific regular expression.
2. Specific python libraries: the "Display message" categories has been built by extracting strings with a python library called "Enchant". Indeed, it provides a function that allows determining if a given word is or not an English word. Based on that I was able to split all words from strings and compute a percentage of English words in a sentence and finally supposed that this string can be a message written by the malware developer. Moreover, the "command line" categories have been build by using the "spawn" function from the distutils python library in order to determine if a given is a known Linux command line or not.
3. Linux command lines: When python libraries were insufficient to sort strings into specific categories I executed Linux command lines through the malware strings analyser script. Indeed it is the case for the Symbols and command lines categories. Indeed, command lines and symbols strings found by executing and then parsing the output of respectively the "command" and "readelf" [28] Linux command lines.
4. Dictionary lists: the Identifiers category is an only based on real user inputs. Consequently, strings from this category have been extracted by using two lists of usernames and passwords builds by using connection attempts data from Cowrie database.

By using those four techniques, the malware strings analyser is able to create a table containing for each raw strings, the interesting part of it and its type (name of the associated category). For all other strings that don't match to a known category, they are noted as the "other" category. Furthermore, it is important to notice that the malware string analyser execute all of these sorting function in a specific order. Indeed, each function browses all extracted raw strings and remove from the list each string detected as belonging to a known categories. Consequently, an optimise sequence has been built to obtain more reliable results. For more information, the code of these sorting function is stored in the parser.py python file in available on the malware string analyser GitHub [26].

3.2.3 Virus total analysis

The malware strings analyser scripts are in fact focused on strings extracted from potential malware binaries, collected by Cowrie honeypot. However, in order to confirm that those binaries were known malware, I decide to apply an external malware detection analysis called Virus Total [9]. This web service provides a large database of known malware, supplied by all users that upload their suspect files into their website. It is possible to upload a file on their platform to be analysed. Then, Virus Total will compare this file to their database by using for instance Hashes of the analysed file. Indeed, a hash is a mathematical algorithm that will generate a unique string which corresponds to the print of a given file. Multiple algorithms can be used to perform this tasks with Virus Total, but the currently allowed hash functions are MD5, SHA1 and SHA256. However, a hash is supposed to be unique, consequently, malware developers can introduce some minor modification in their source code in order to modify the hash without modifying its behaviour. By using this Hash virus total will return to the user data generated by a static analysis of this file if it has been already analysed. Moreover, it will submit it to a list of common antivirus and provide a list of their results.

Furthermore, to automate the usage of this powerful tool, Virus Total team have developed an API (Application Programming Interface), which allow developers to send HTTP requests to their tool automatically from a program. This API is accessible by using an API key generated during the creation of a Virus Total account. Those API key can be public or private and give an access to more or fewer functionalities. Despite the limited number of functionalities in the public API compare to the private API, the real disadvantage of the public API is that the number of requests is limited in the time. Indeed, users are restrained to a small number of requests per minutes and days. This could be an issue for my research due to a large amount of binary to analyse automatically during my research. Consequently, I decided to contact the Virus Total team that gave me a private API key in view of my project research.

Hence, by using this Virus Total API key, the malware strings analyser provides a Virus Total options allowing to perform a global analysis of the file, before the strings analysis part. This has been useful to classify extracted strings depending if they were from a known or a potential unknown malware. Indeed, the aim of the malware string analyser tool is to generate a database containing strings sorted into categories from a given binary, but also to collect global data of this file. Indeed, this will lead to apply data mining techniques from those gathered data, in order to determine if a file is potentially a malware or not. Consequently, global binary information collected during this steps are essential for future analysis.

Then, after the execution of the strings analysis steps, strings sorted as URLs and IP addresses are analysed by using the Virus Total API. Indeed, Virus Total allow its users to submit URL and IP addresses in order to determine if they are known as malicious activities. Consequently, the malware strings analyser tool send requests for each strings define as a URL or an IP address and provide the results in its output. Returned information can allow determining if a found URL in a binary is known as a malware binary host, or if an IP address is known as a command and control server for malware.

4 Honeypot results

4.1 Attacks overview

The Cowrie honeypot has been put in place the 16th March 2016 and will continue to run until the end of the dissertation, the 12th September 2016. During this period a number of 82388 SSH login attempts have been recorded. Those unauthorised connections were detected as being executed from 2706 distinct source IP addresses. However, some of those IP addresses have been more active than others. Indeed, 20% of all of those connections have been performed by 10 distinct IP addresses (see Figure 4). Moreover, attacks from these 10 IP addresses were from a different part of the world such as China, United States, France, Japan and Republic of Korea (see Figure 5). Hence, this list of country is finally the top five of connections per country during this period of attacks on the honeypot.

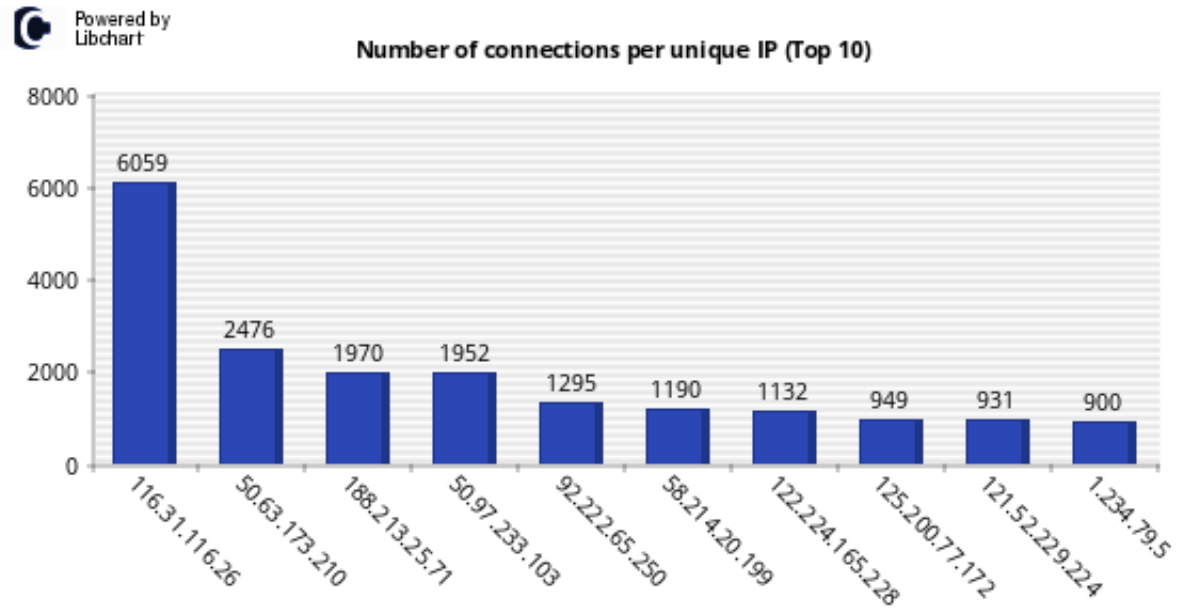

Figure 4: Number of connections per unique IP (TOP 10) - Kippograph

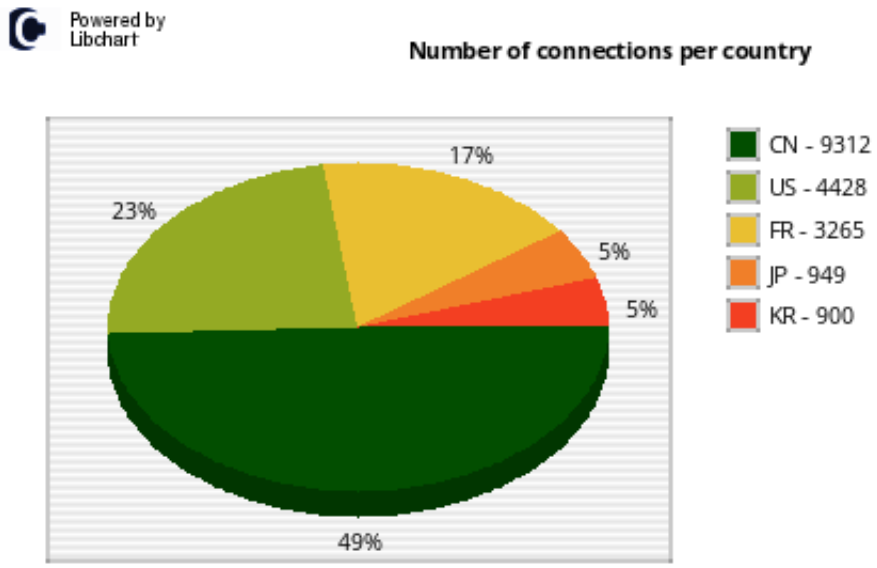

Figure 5: Number of connections per country - Kippograph

Furthermore, the cowrie honeypot has been also able to determine the version and type of the SSH connection library and tool use to initiate and SSH connection on our server. Hence, we can see that attackers use mostly the SSH-2.0 libssh in multiple version such as the 1.6.0 or 1.4.3. These libraries can be used in multiple operating systems such as Linux and Windows. In the same way, the honeypot has detected that in the top 10 of ssh clients the Putty software has been used. This tool originally developed for Windows is now available on Linux systems. Consequently, this information cannot be reused to determine the operating system of the attackers I had hoped.

Kippo graph has also collected all shell inputs written by attackers once they were connected. Concerning this post-compromise part, a total of 60023 commands has been inputted with only 1011 distinct commands. Those data are really interesting because the top 10 of most commonly used commands represent 49% of all detected inputs (see Figure 6). The first command in this ranking is "mkdir /tmp/.xs/" which consists of creating a hidden directory into the temporary directory of the targeted Linux system. This command has been used a total of 10875 times, consequently, we can easily suppose that this has been executed by a bot. This time of script consists of automating the execution of given command lines. In the same way, we can see that the top 5 of failed command, which are all invalid shell command lines, represent a total of 10834 commands. Consequently, it is difficult to believe that a human attacker has been able to make this amount of mistakes. It is more likely that this has been performed by a script containing an error.

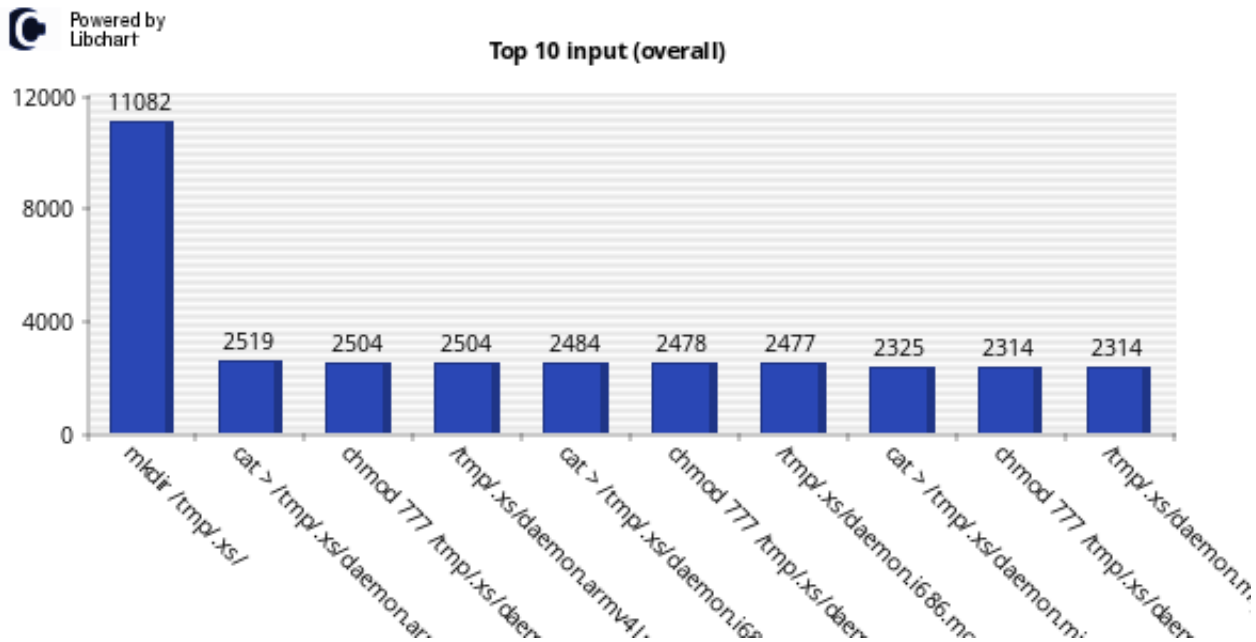

Figure 6: Top 10 input (overall) - Kippograph

4.2 Authentication attempts

On the 82388 attackers SSH login attempts recorded, 66% of them were considered as successful connection because a full shell access was granted to them (see Figure 7). This step of connection attempts on our SSH server can be considered as a brute force attack. Indeed, attackers through their scripts tried successively a list of common usernames and passwords in order to access the server remotely.

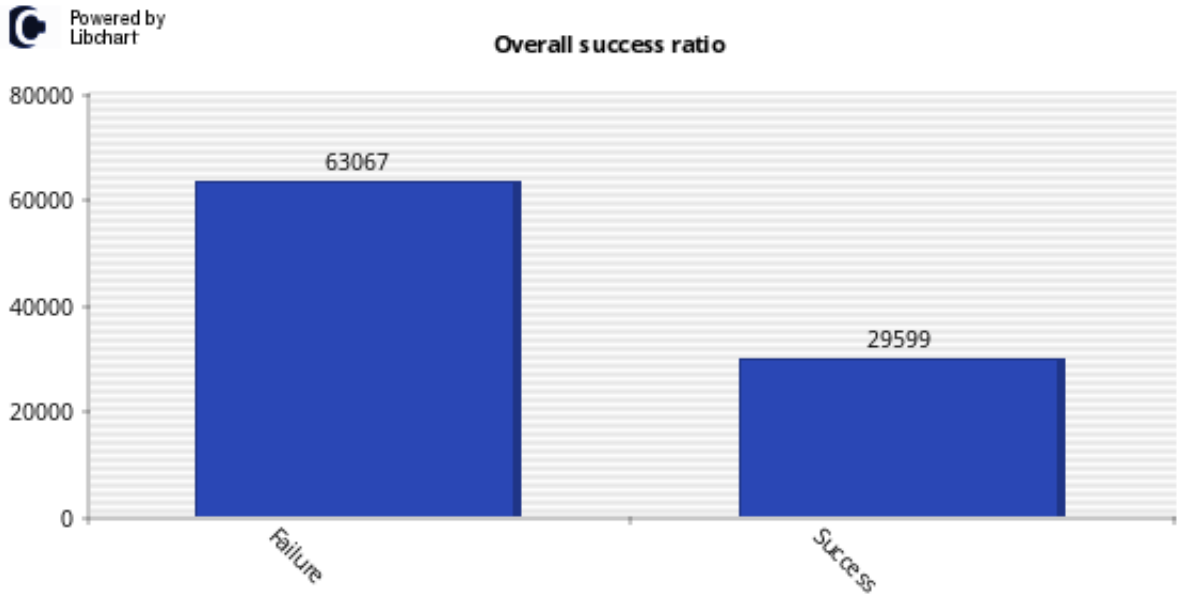

Figure 7: Overall success ratio - Kippograph

This part has been really interesting to analyse because the cowrie honeypot as collected and stored all of those identifiers in its database. Those data have allowed the malware strings analyser to extract known identifiers from

hard coded strings on malware binaries. Moreover, these data confirm relevance to use complex passwords other than simple common strings such as "admin", "root" or "12345". Indeed these last 3 passwords are the top 3 of tested passwords on the SSH server. In the same way, usernames are also important to choose, indeed we can see that attackers try default usernames (and also default passwords) from known networks devices such as "git", "oracle" and "postgres", which correspond to known tools and databases (see Figure 8).

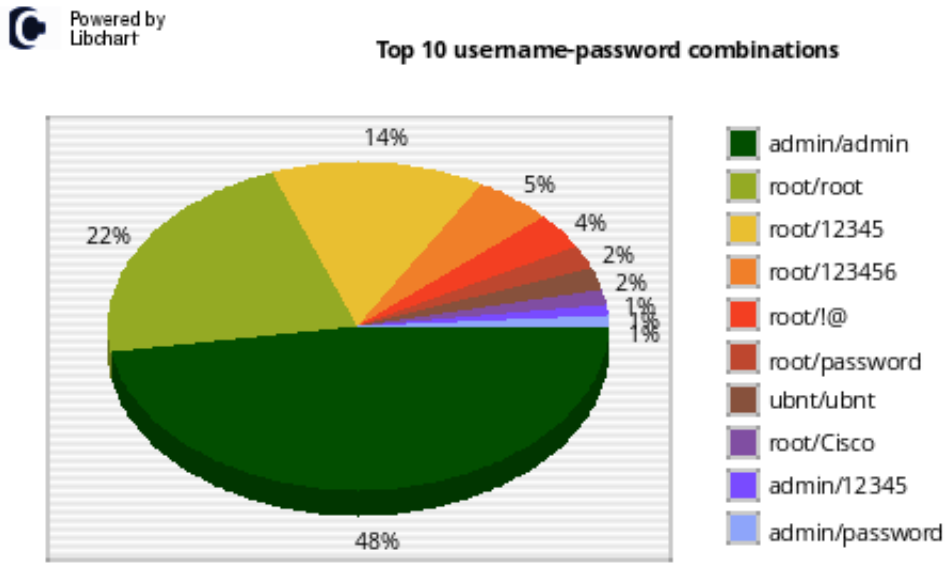

Figure 8: Top 10 username-password combinations - Kippograph

Finally, we can observe, that the brute force attack here consists to use a short list of known identifiers and not try to modify character by character in order to find the correct username/ password. Indeed, attackers only try a fixed number of identifiers by IP addresses before achieving to access to our SSH server. Consequently, we can easily suppose that attackers only try to access to a server with weak authentication identifiers standards and change to another if usernames and passwords are not in their dictionaries. This allows attackers to browse all public addresses one by one and testing in case of an SSH port open all of their passwords. This attack is really simple, however, the probability to obtain an access to a server by this the technique is not low because of the large range of public IP addresses available.

4.3 Malicious downloads

Once attackers were logged on the server through SSH, a common task was to download what we can call a malware deployment script. Indeed, attackers usually download a shell script by using mostly the "wget" command line, which is used to get a content from a given URL, in our case a script. The advantage of this script is that it could be run on every Linux server without downloading or installing additionals packages. Then, the attackers run this shell script that will download binaries with the wget command, grant execution rules to the binary by using the "chmod" command line and run it. This sequence of command is re-executed for each compiled version of the malware.

This way to compromised a Linux is surprisingly really simple. Indeed, once the script is downloaded it will download and run each malware, compiled for most common processor architecture in order to be able to correctly run. However, the script does not try to get more information of the system in order to download and run the appropriate malware.

All of these downloaded versions of malware were those that I chose to analyse with the malware strings analyser script. I decided to re-download them manually from each malware deployment scripts downloaded by attackers on my server. I had to modify these script shell manually to only keep the downloading part. However, I was not

able to directly perform this task just after the attack because of the random hour when those events occurred. Consequently, when I try to re-download all binaries from attackers scripts some URLs pointing to binary to download had expired. This shows us that attackers have multiple servers used to host their malware and deployment scripts, but their usage expired probably in order to avoid to be easily tracked.

4.4 Malware binaries analysis

During the entire period of the project research, I have collected a large sample of malware binary. All of them were known malware and were usually the same. Indeed, the majority of them were detected as the malware called "GayFgt". This has been possible to be detected by using the Virus Total option of the malware strings analyser script (see Figure 9).

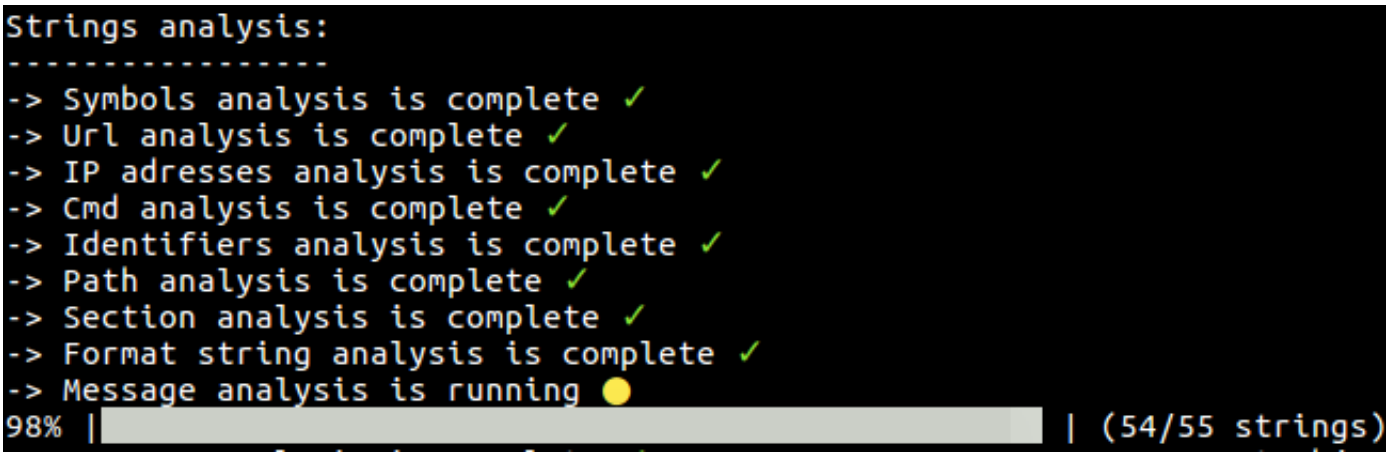

Figure 9: Positive malicious file output - Malware Strings Analyzer

Furthermore, the larger part of the analysis was not focused on a full static or dynamic analysis of the malware as usual. Indeed, I decided to investigate on the content of strings hard coded in those malware binaries. First of all, it is important to recall that malware can be packed, which consists of encrypting a the malware. Then, once it is executed the malware will decrypt its code itself dynamically and run normally. The current version of the malware strings analyser can detect that it has been packed but it will abort the execution and it does not try to unpack it with known unpacking techniques (for instance UPX packing method). In the same way, some collected binaries were stripped which results to loose strings containing symbols information about the malware. As for the packing issue, the first version of the malware string analyser will abort its analysis in the case of strip malware.

Concerning the analysis of not stripped and packed malware, the malware strings analyser has been able to extract some useful information about each malware. I have been able to collect some other URLs and IP addresses stored in this malware. Indeed, found URLs were used to download the new version of the malware from a remote server. This has been confirmed by using the Virus Total option of the malware strings analyser which analysed each URLs found and confirmed that those URLs were known as storing malicious content (see Figure 10). Furthermore, IP addresses found have been detected has remote command and control server by using the same Virus Total option of the malware strings analyser tool.

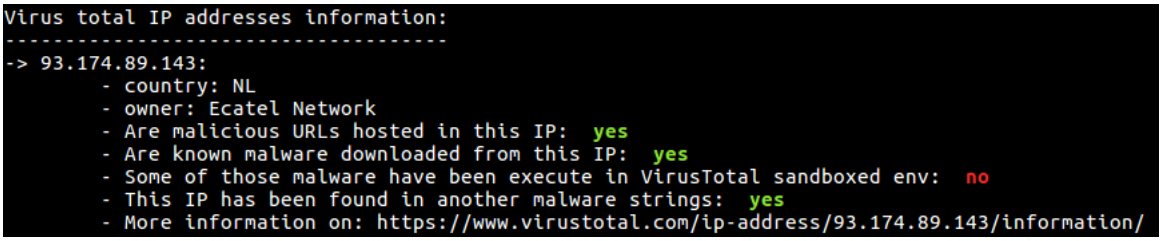

Figure 10: Positive malicious IP address output - Malware Strings Analyzer

Finally, other interesting categories were the potential displayed message and format strings. Indeed, each of them contains strings that explicitly talk about TCP and UDP Flooding, followed by a format string making think of an IP address. This and other strings resembling a list of remote commands and error message, suggests that these malware were connected to a remote command and control server in order to perform DDoS attacks with TCP and UPD packets against a given targeted IP address. Some specific research concerning the "GayFgt" malware have confirmed that the DDoS attacks was one of its functionality and confirmed my assumption by using only a static string analysis.

5 Future research

As shown in the previous sections, the Cowrie honeypot allowed me to collect a sample of malware, currently used to compromise and manipulate remotely infected Linux servers. Based on these gathered malware binaries, I have been able to develop a script, called "malware strings analyser" (MSA), and I use it to extract, sort and store hard-coded strings from a given malware binary into a database. Moreover, as seen previously, for each analysed malware, a table is created containing all strings sorted into their types (10 strings categories).

Furthermore, this works as to be continued to populate a large database containing sorted strings of a maximum of binaries detected as malware. Simultaneously, another database, respecting the same structure as the previous one, has to be created. This database will store hard-coded strings of non malicious binaries, by using the malware string analyser script. The aim of this task is to create two distinct groups of extracted data from malicious and benign binaries.

The second step will be to extract and define patterns that represent a malicious binary, only by using their previously extracted and sorted strings. This can be performed in two manners: by supervised and unsupervised techniques. The difference between the both is that the supervised techniques consist, in our case, to manually define which binary is or not a malware. Then, by using a machine learning algorithm such as the C4.5 (based on the ID3 algorithm) [29] a decision tree is to be created in order to determine, for a given sample of sorted strings, if a given binary has a certain percentage of chance to be malicious or not.

In order to improve this decision tree, the definition of patterns and the usage of the machine learning algorithm steps have to be repeated in order to adjust the number of questions (nodes of the decision tree) that will generate the result.

This decision tree will be used as a pre-analysis binary tool, allowing malware analysts to perform a first and fast classification of a given unknown binary. This could be automated by using a honeypot that automatically collect, export malware binary and populate the malware strings databases by using the malware string analyser tools. Then, it could be possible to continuously update and improve the decision tree to finally provide a web interface or an API. These user interfaces, could be based on the Virus Total model, allowing every malware researcher to submit their binary to evaluate the chance that a binary could be a malware.

6 Concluding discussion

In this paper, we presented the usage and the installation of an SSH honeypot. As we have seen this tool allow to easily collect basics attacks on Linux server quickly without using a lot of resources. Indeed, only the cost is the server which is currently not really expensive especially if we choose a Virtual machine as a server. However, it is important to notice that honeypot is a known technique by attackers and given the fact that honeypots are software it is impossible to be sure that they are safe to deploy on a network or on a server containing sensitive data. That is why I choose the Cowrie honeypot which is a maintained project with an available code source which reduces the risk of vulnerability, even it is impossible to be sure that there is no security hole on it.

By using this honeypot I have been able to analyse and understand attackers behaviours that attack Linux Server through the SSH port. I was mainly surprise of the simplicity of their attack scenario. Indeed, some basic security standard can avoid the totality of intrusions that only apply a basic brute force attack, with login attempts by using most commonly used usernames and passwords. First of all, it is possible to change default usernames and choose strong passwords with alphanumerics characters in upper and lower case associate to symbols. Moreover, it is

possible to install services on the server such as fail2ban, which consists of blocking after a configured number of fail login attempts. Finally, it could also be a good practice to change the SSH port, initially port 22 to another.

I have also been able to collect and analyse a sample of malware currently used by attackers. Based on their number I could not be able to perform a specific analysis for each of them. Consequently, I started to use Cuckoo malware analysis framework, however, this tool was mainly focused on Windows malware analysis. Despite the usage a GitHub project that improvement Cuckoo in order to analyse Linux malware, I was not able to configure it correctly. Hence, I used Limon another malware analysis tool but focused on Linux malware. I applied it on the first collected malware, but the quantity of generated data was to large, due to its static and dynamic analysis. However, in its report, I found interesting to analyse extracted strings. Based on those strings I developed the malware strings analyser tool with the goal to be re-used in order to find patterns on those strings that can be crossed to a data mining algorithm. This could improve the malware pre-analysis techniques currently used.

Finally, in the way as the HoneyNets project, all of the collected data are aimed to update known attackers techniques and improve knowledge about Linux server security in order to generate adapted security protection against those specific attacks. Indeed, this project contributes to the infinite race between attackers and security researchers, in which attackers are still one step ahead of us.

7 Bibliography

[1] Canonical. [Online]. Available from: http://www.ubuntu.com/ [Accessed 7 September 2016].

[2] Debian. [Online]. Available from: https://www.debian.org/ [Accessed 7 September 2016].

[3] R. Berthier, J. Arjona, M. Cukier (2009). Analyzing the process of installing rogue software. [Online]. Available from: IEEE Xplore. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5270293. [Accessed 5 March 2016].

[4] Valli, C., Rabadia, P., Woodward, A. and Research Online (2015). Patterns and patter - an investigation into SSH activity using Kippo Honeypots. [Online]. Available from: http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1128&context=adf [Accessed 19 June 2016].

[5] micheloosterhof (2016). Micheloosterhof/cowrie [Online]. Available from: https://github.com/micheloosterhof/cowrie [Accessed 7 September 2016].

[6] Common vulnerabilities and exposures (CVE) (1999). [Online]. Available from: https://cve.mitre.org/ [Accessed 7 September 2016].

[7] L. Spitzner (2003). The Honeynet Project: Trapping the Hackers. [Online]. Available from: IEEE Xplore. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1193207 [Accessed 5 March 2016].

[8] R. Berthier, J. Arjona, M. Cukier (2009). Analyzing the process of installing rogue software. [Online]. Available from: IEEE Xplore. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5270293. [Accessed 5 March 2016].

[9] Free online virus, Malware and URL scanner (n.d.). [Online]. Available from: https://www.virustotal.com/ [Accessed 7 September 2016].

[10] G. Salles-Loustau, R. Berthier, E. Collange, B. Sobesto, M. Cukier (2011). Characterizing Attackers and Attacks: An Empirical Study. [Online]. Available from: IEEE Xplore. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6133079. [Accessed 5 March 2016].

[11] madirish (2015). Madirish/kojoney2 [Online]. Available from: https://github.com/madirish/kojoney2 [Accessed 7 September 2016].

[12] An Introduction to Linux-based malware (n.d.). [Online]. SANS Institute. Available from: https://www.sans.org/reading-room/whitepapers/malicious/introduction-linux-based-malware-36097 [Accessed 19 June 2016].

[13] LukdurfinaJakub, Koustek, P. and Zemek (n.d.). Psyb0t Malware: A step-by-step Decompilation case study. [Online]. Available from: https://www.computer.org/csdl/proceedings/wcre/2013/9999/00/06671321.pdf [Accessed 19 June 2016].

[14] Monnappa, K. A. (n.d.). Automating Linux Malware Analysis Using Limon Sandbox. [Online]. Available from: https://www.blackhat.com/docs/eu-15/materials/eu-15-KA-Automating-Linux-Malware-Analysis-Using-Limon-Sandbox-wp.pdf [Accessed 19 June 2016].

[15] Yan, W., Zhang, Z. and Ansari, N. (2008). Revealing packed Malware. [Online]. IEEE. Available from: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4639028 [Accessed 19 June 2016].

[16] Mulliner, C., Liebergeld, S. and Lange, M. (2011). Poster: HoneyDroid -creating a Smartphone Honeypot. [Online]. Available from: http://www.ieee-security.org/TC/SP2011/posters/HoneyDroid__Creating_a_Smart_Phone_Honeypot.pdf [Accessed 7 September 2016].

[17] The Honeynet project (2016). [Online]. Available from: https://www.honeynet.org/ [Accessed 7 September 2016].

[18] honeynet (2010). Honeynet/honeysnap [Online]. Available from: https://github.com/honeynet/honeysnap [Accessed 7 September 2016].

[19] desaster (2015). Desaster/kippo [Online]. Available from: https://github.com/desaster/kippo [Accessed 7 September 2016].

[20] Web hosting, cloud computing and dedicated servers- OVH (2016). [Online]. Available from: https://www.ovh.com [Accessed 7 September 2016].

[21] GitHub (2016). Build software better, together [Online]. Available from: https://github.com/ [Accessed 7 September 2016].

[22] MySQL (2016). [Online]. Available from: http://www.mysql.com/ [Accessed 7 September 2016].

[23] ikoniaris (2016). Ikoniaris/kippo-graph [Online]. Available from: https://github.com/ikoniaris/kippo-graph [Accessed 7 September 2016].

[24] the Apache HTTP server project (1997). [Online]. Available from: https://httpd.apache.org/ [Accessed 7 September 2016].

[25] Nginx (2009). [Online]. Available from: https://nginx.org/ [Accessed 7 September 2016].

[26] Yann Ferrere (2016). Ferrery1/ProjectResearch Malware Strings Analyzer [Online]. Available from: https://github.com/ferrery1/ProjectResearch/tree/master/utils/malwareStringsAnalyzer [Accessed 7 September 2016].

[27] SQLite (n.d.). [Online]. Available from: https://www.sqlite.org/ [Accessed 7 September 2016].

[28] inux man pages: Readelf (1) (n.d.). [Online]. Available from: http://www.manpages.info/linux/readelf.1.html [Accessed 7 September 2016].

[29] Hssina, B., Merbouha, A., Ezzikouri, H. and Erritali, M. (n.d.). A comparative study of decision tree ID3 and C4.5. [Online]. Available from: http://saiconference.com/Downloads/SpecialIssueNo10/Paper_3-A_comparative_study_of_decision_tree_ID3_and_C4.5.pdf [Accessed 7 September 2016].

8 Appendices

Overall honeypot activity

Total login attempts	92683

Distinct source IP addresses	2787

Active time period	
Start date (first attack)	**End date (last attack)**
Thursday, 17-Mar-2016, 08:59 AM	Thursday, 08-Sep-2016, 00:09 AM

Figure 11: Overall honeypot activity - Kippograph

Overall post-compromise activity

Post-compromise human activity	
Total number of commands	**Distinct number of commands**
62117	1068

Downloaded files	
Total number of downloads	**Distinct number of downloads**
2711	451

Figure 12: Overall post-compromise activity - Kippograph

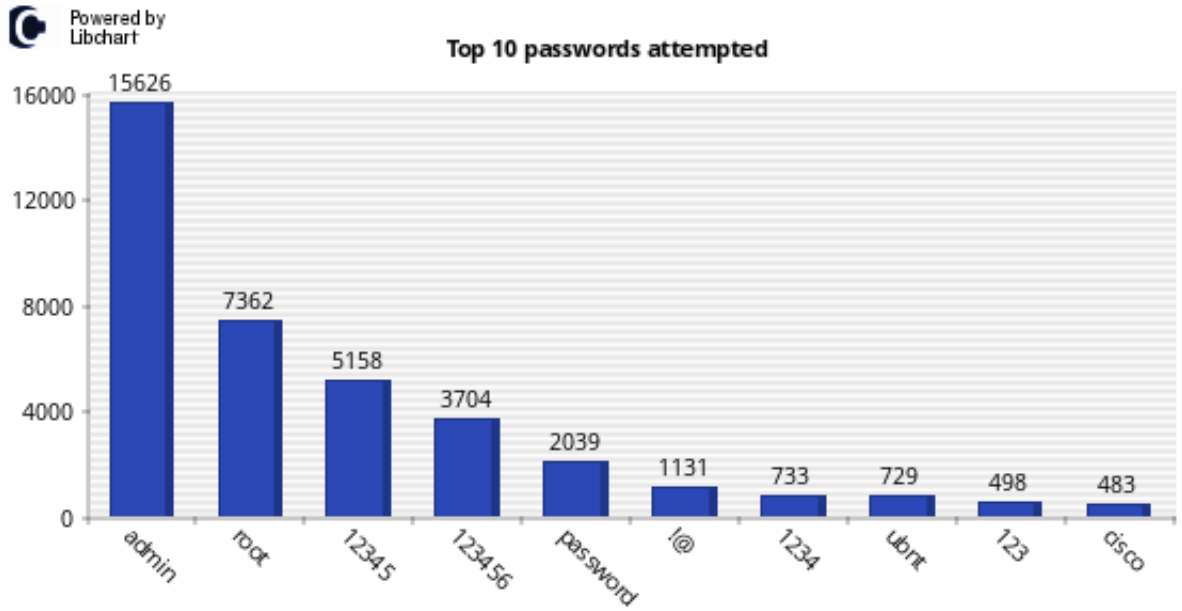

Figure 13: Top 10 passwords - Kippograph

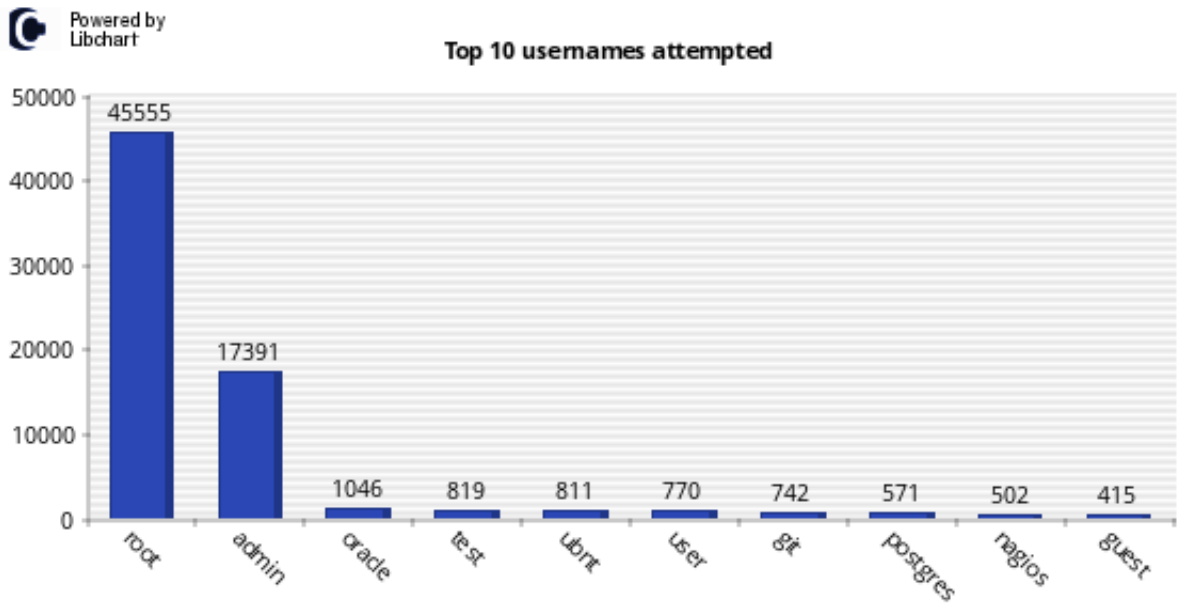

Figure 14: Top 10 usernames - Kippograph

Figure 15: Most successful logins per day - Kippograph

Figure 16: Successes per week - Kippograph

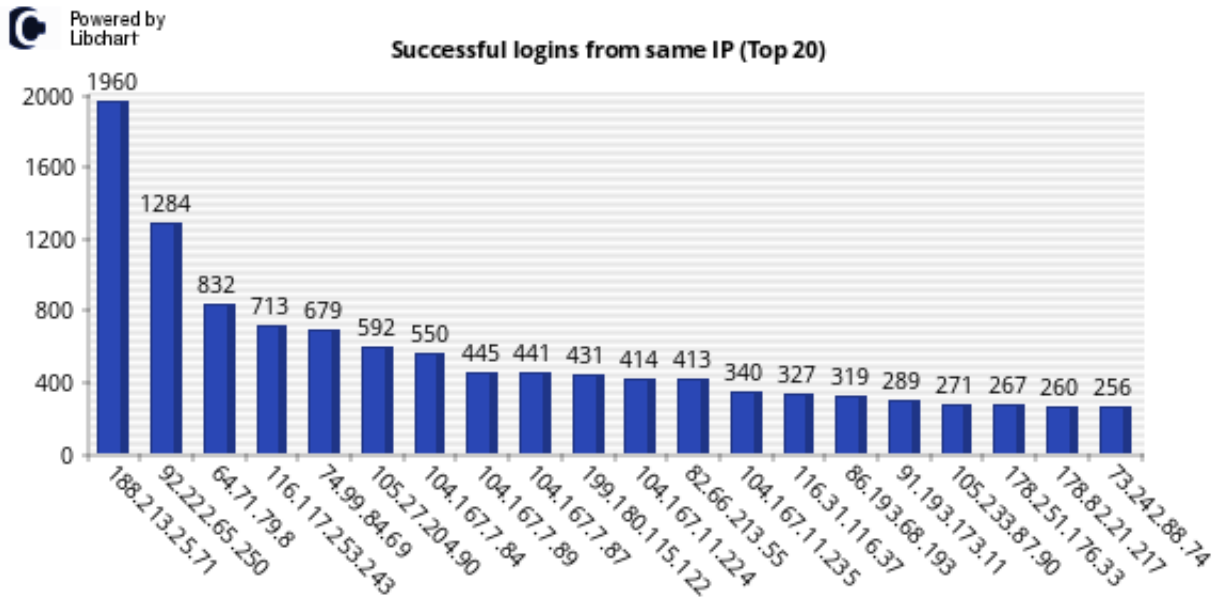

Figure 17: Successful logins from same IP (TOP 20) - Kippograph

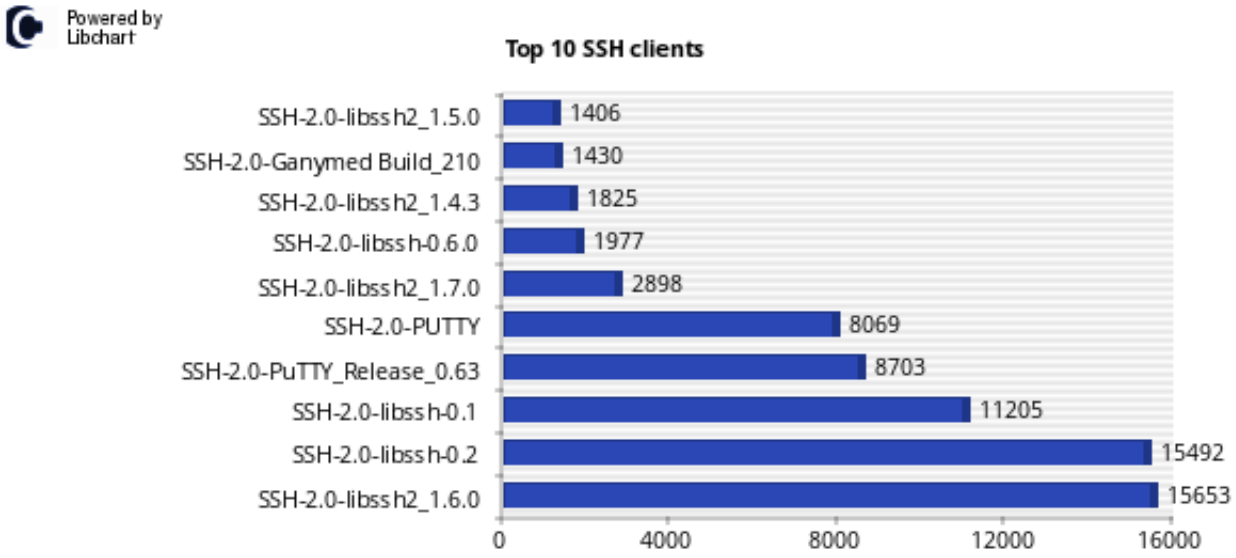

Figure 18: Top 10 SSH clients - Kippograph

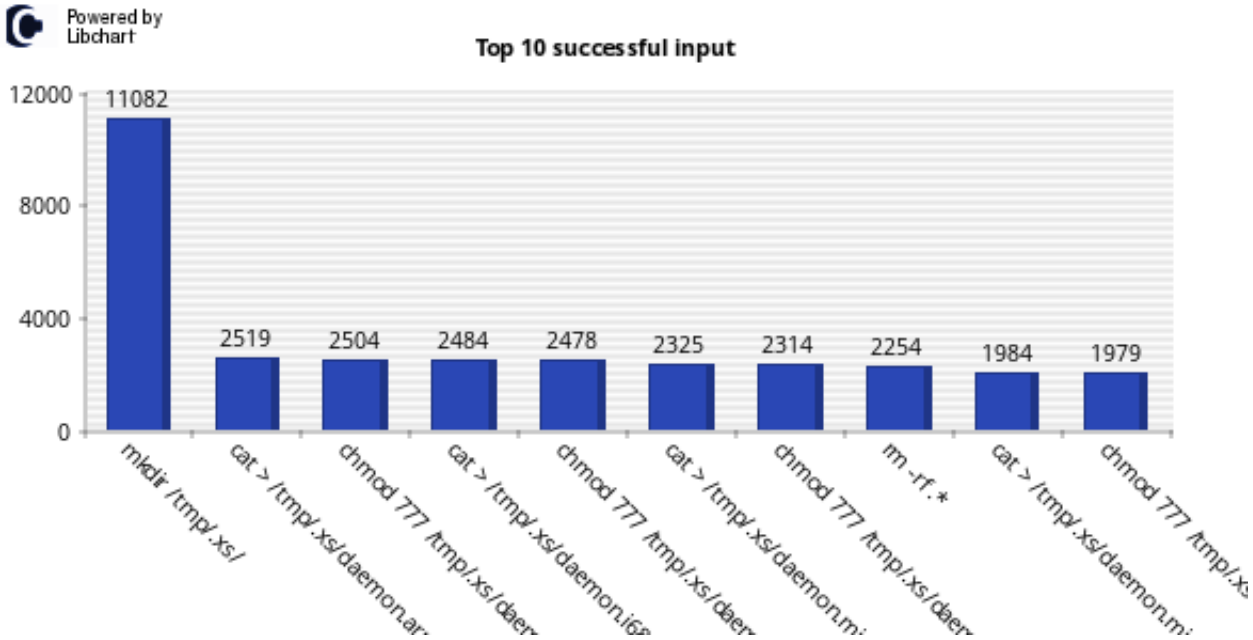

Figure 19: Top 10 successful input - Kippograph

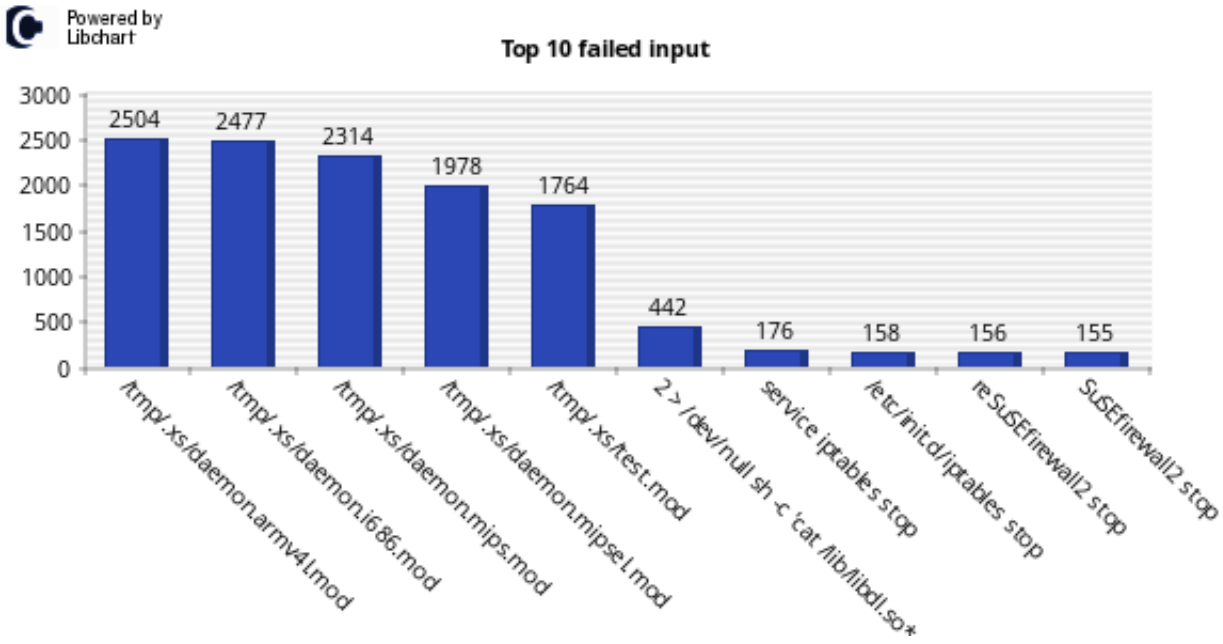

Figure 20: Top 10 failed input - Kippograph

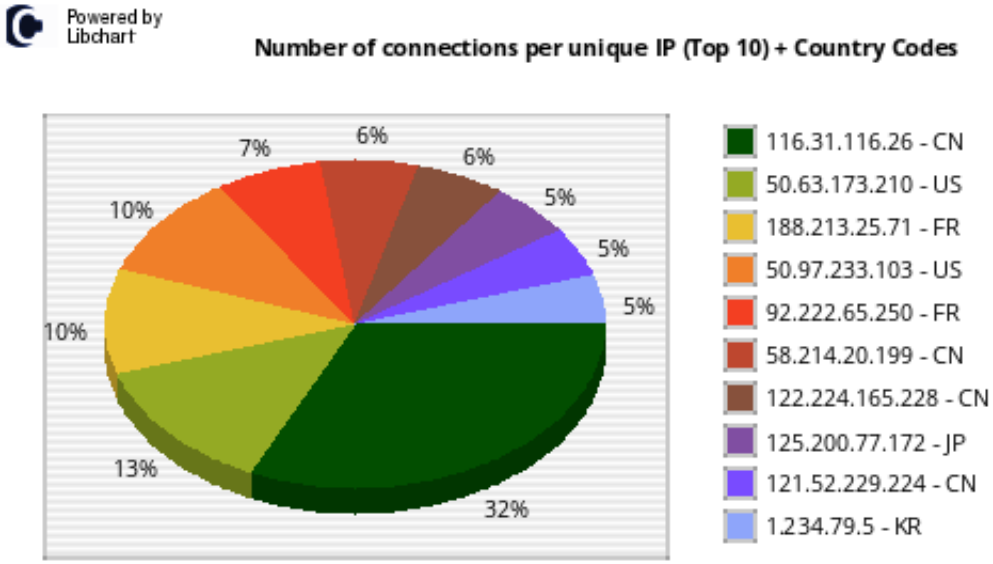

Figure 21: Number of connections per unique IP (Top 10) - Kippograph