University of Kent

A systematic analysis of attacks on a honeypot

CO880 - Project and Dissertation

Yann Ferrere
ID 15906875
yf57@kent.ac.uk

Abstract

Acknowledgement

Table of contents

List of Figures

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 computer language and are hosted on a server to store users data and to be accessible remotely.

However, all of these communication 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 analyze what are the steps of an attack on a server. Moreover, servers work as common computers and need to have an operating system 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 or Debian.

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 analyze 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 bruteforced 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 72500 SSH login attempts with 62 percent of successful connection. Those attacks were from 2550 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 technic to collect attacks on a Linux server is not new. Several papers have already put in place this configuration in order to analyze attackers behaviours. However, there are plenty of ways to implement a honeypot. Some papers have created their own network architectures with multiple machines, while others have used existing solutions such as Kippo. In this dissertation, the Honeypot is cowrie, a fork of the kippo project which is not longer maintained. Consequently, this paper shows in a real case scenario that this honeypot can be an appropriate choice in 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). 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 in order to collect data and analyze security issues. Consequently, all along this project research, a databases 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 analyzing 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, extract, 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 project already performed is exposed first. Then, the methodology of my research will be explained.

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 analyzer will be shown.

A concluding discussion will summarize 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" 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 " 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 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 " 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 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", 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 " 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 ", 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 ", 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 Methodology

4 Implementation and configuration

4.1 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. 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.

4.1.1 Honeypots overview

As seen during the literature and related works review, Honeypots are not a recent technics 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. Moreover, the HoneyNet Project 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, 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 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.

4.1.2 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 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, 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, 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 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.

4.1.3 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 were 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, its 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), 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 number of login attempts and distinct source IP addresses, easy to understand and interpret.

The installation is also simple and fast to implement. First of all, Kippograph is a web interface consequently a webserver such as Apache or Nginx 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 localization (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.

4.2 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), 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.

4.2.1 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, sort them depending on their groups and store them in an SQLite database. 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.

4.2.2 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 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" 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 technics 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 browse 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.

4.2.3 Virus total analysis

5 Analysis and results

5.1 Attacks overview

5.2 Authentication attempts

5.3 Malicious downloads

5.4 Malware binaries analysis

6 Future research

7 Concluding discussion

8 Bibliography

9 Appendices