UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Laurea Triennale in Informatica

Analysis of network traffic on the public network through a honeypot

Relatore:

Prof: Luca Deri

Prof: Fabrizio Baiardi

Candidato:

Sergio Garrido de Castro

ANNO ACCADEMICO 2022/2023

Acknowledgments

First and foremost, I want to express my gratitude to my parents, who have provided me with their effort and sacrifice, giving me the opportunity to receive a quality education and become a Computer Engineer.

To my friends from Medina del Campo, my lifelong companions, who have been there with me from the beginning until this point in my life.

To my friends from Colegio Mayor Belardes, who have accompanied me through the best and worst times and have helped shape me into the person I am today throughout all these years.

To my friends from Computer Engineering, without whom these years of study would not have been the same. They have proven to be extraordinary travel companions, sharing experiences, memories, and knowledge.

To all of them, I am grateful for their support, the moments we have shared, the invaluable help they have provided, and the life lessons they have taught me. I am proud of them, and I couldn't be more thankful.

Abstract

The objective of this project is to investigate honeypots with the intention of configuring and developing new functionality in order to monitor network traffic and determine the distribution of traffic in the network and the presence of malicious activity in such traffic.

The project has research undertones. A honeypot is a security tool that can be dangerous if used incorrectly or unethically. The honeypot used will be employed in a controlled environment with all possible precautions to try to ensure its proper functioning."

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Objectives and Expected Results

The main objective of this project is to collect data from a public network using a selected honeypot based on certain criteria. The collected data will be related to the connections made within that public network.

Once the data is collected, it will be analyzed using analysis techniques and appropriate tools to distinguish the types of connections made and the protocols used (such as SSH, DNS, or TFTP). Furthermore, the analysis will take into account whether there were any downloads during the connection, the remote IP address, and the port, in order to identify any suspicious activity.

The connections in a public network will be monitored over a specific period of time, typically between 20 and 30 days, to obtain a significant and representative sample of the network's activity. The collected data will be stored in a log file, and the corresponding analysis techniques will be applied to that log file. The expected results in the log files will be the detected connections based on the predefined configuration of the honeypot. If the honeypot is configured to detect only specific protocols, then the log file will contain only the connections that were detected using those protocols. The results will also depend on the configuration and usage mode of the honeypot. Different honeypots may yield different results due to variations in their level of interaction and network emulation.

In summary, the results collected in the log file after monitoring a public network over a period of time will consist of incoming connections that meet the predefined configuration conditions of the honeypot. Once these connections are collected, they will be analyzed and classified. It is expected that the detected connections will primarily involve common protocols, such as HTTP, HTTPS, and DNS, which should represent a significant portion of the monitored connections. File transfer protocols, such as TFTP or FTP, are also expected to contribute to a significant

portion of the traffic. Protocols involving remote connections, like SSH, are expected to be less prevalent in proportion. Traffic related to SQL database protocols is not anticipated since there are no plans to implement any databases.

Chapter 2

Theoretical Foundations of a Honeypot

A honeypot is a software that serves as a security tool deployed within a network with the purpose of being the target of a potential cyber-attack, enabling its detection and gathering information about the attacker and the attack itself.

Its main and most notable characteristic is that honeypots are not only designed to defend against attacks, but they also act as invisible decoys, allowing the detection of potential attacks before they can impact other critical systems. However, honeypots can be designed for various objectives, such as alerting the presence of an attack, gathering information about an attack without interfering with it, or slowing down the attack to protect the rest of the system. Based on this, honeypots have different use cases, but their primary purpose is to distract potential attackers from more important information and machines within the actual internal network. They also serve to learn about the types of attacks that can be encountered and examine those attacks during and after the exploitation of the honeypot tool.

2.1 Definition

The term "honeypot" refers to a software tool that, literally translated from English, means "honey jar." However, in the context we are interested in, this word has a different connotation, alluding to a "honey trap." This definition originated in the field of espionage, where it describes how spies, like Mata Hari, used romantic relationships to obtain secrets, thus setting up a "honey trap." Enemy spies often fell into these traps and were then blackmailed into revealing all the information they possessed.
In the field of computer science, an analogy is drawn with this concept, as a honeypot, as known today, operates similarly by luring attackers into a trap. According to this definition, a honeypot is a computer system specifically designed to act as a decoy, sacrificing itself to attract cyber-attacks. In summary, a honeypot is a

monitored computer resource used to test its ability to resist or be compromised by cyber-attacks. Figure 2.1 illustrates a system with a honeypot in place.

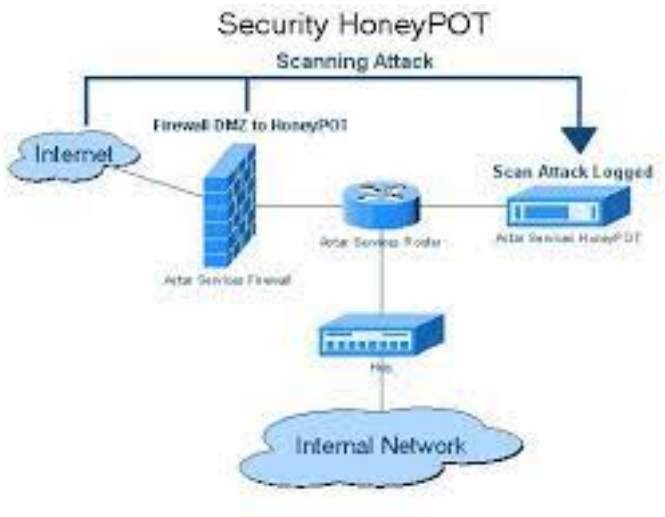


Figure 2.1: Honeypot Architecture

2.2 Types of Honeypots

Within honeypots, there are several classifications [1] to distinguish them. The first classification we can offer is based on the type of implementation:

- Physical: This refers to a real machine with its own IP and MAC addresses. The machine simulates system behaviors. Implementing physical honeypots can be costly as it requires acquiring new machines and configuring specific hardware.
- Virtual: A virtual honeypot [2] is a simulated machine with emulated behaviors, including the ability to respond to network traffic. Multiple virtual honeypots can be simulated on a single system. This type allows for the installation and hosting of various operating systems within the network. To convince attackers that a virtual honeypot is running a specific operating system, it is necessary to simulate the target operating system's TCP/IP stack in order to deceive them. Another machine is responsible for responding to network traffic directed at the virtual honeypot.

Table 2.1 provides a breakdown of the advantages and disadvantages of these two types:

	Physical Honeypot	Virtual Honeypot
Advantages	Easy deployment of multiple honeypots Scalability and low-maintenance costs Inexpensive Quick and simple implementation Ideal for low-interaction honeypots	Real machine on the network More realistic Harder to identify as a honeypot Ideal for high-interaction honeypots
Disadvantages	More easily detectable by attackers Difficulty in simulating complex systems Collects less data	Expensive Impractical for systems with many IP addresses

Table 2.1: Advantages and disadvantages of physical and virtual honeypots

Honeypots can also be distinguished based on their purpose:

- Research Honeypots: These honeypots are designed to gather information about the specific methods and techniques used by attackers and to detect vulnerabilities in the network and assets. They are usually more complex than production honeypots. They are often used by organizations to better understand security risks, analyze network traffic, and assess the risks present in the network. This knowledge allows them to propose and design appropriate security strategies. Research honeypots are used by corporations, private companies, and individuals.

- Production Honeypots: These are the most common type and are used to gather information and intelligence about cyber-attacks within the production network. The collected data includes IP addresses, date and time of intrusion attempts, traffic volume, and other attributes that may vary depending on the tool used and its configuration. Production honeypots are relatively simple to design and implement but are less sophisticated than research honeypots in terms of the intelligence gathered.

Types of honeypots can also be classified based on design criteria [3]:

- Pure Honeypots: These are complete production systems. The attacker's activities are controlled using a bug tap, which is installed on the honeypot's network link. No additional software installation is required for operation. Pure honeypots are useful, but the stealth of their defense mechanisms can be ensured with a more controlled mechanism.

- High-Interaction Honeypots: These honeypots are designed to engage attackers for extended periods by simulating the activities of a production system

hosting a variety of services and presenting a network of targets such as different databases. The extended engagement time allows for the collection of a larger amount of data, providing greater insights into attack methods, techniques, and even the identity of the attacker. They consume more resources but also provide more relevant and higher-quality data. A honeypot perimeter or honeywall should be configured around the high-interaction honeypot, offering a single entry and exit point to monitor and manage all network traffic, preventing lateral movements from the decoy system to the real system. High-interaction honeypots offer more security but are also more difficult to detect. Multiple honeypots can be hosted on a single physical machine using virtualization, allowing for quick restoration if a honeypot is compromised. If virtual machines are not available, a separate physical machine must be used for each honeypot, which can be cost-prohibitive.

- Low-Interaction Honeypots: These honeypots use minimal resources and collect basic information from attackers. They are easy to configure and maintain but are less effective. They only simulate the specific services requested by attackers. Due to their low resource consumption, several virtual machines can be hosted on a single physical system without difficulty. Virtual systems have short response times and require less code, reducing complexity. As mentioned earlier, most production honeypots are low-interaction honeypots.

- Sugarcane Honeypots: This type of honeypot masquerades as an open proxy. It can take the form of a server designed to resemble a misconfigured HTTP proxy.

- Deception Honeypots: This is a proactive cybersecurity measure based on artificial intelligence, machine learning, and other advanced technologies to enhance and automate data collection and analysis.

There is also a final classification type based on the type of activity detected:

- Email or Spam: The honeypot implements a fictional email address in a hidden field that can only be detected by an automated address harvester or site crawler. Since the address is not visible to regular users, all correspondence sent to that inbox can be categorized as spam. The organization can then block the sender's IP address as well as any message matching the blocked message's content.

- Databases: The honeypot decoy is intentionally designed to appear as a vulnerable and fictitious dataset. This decoy helps companies monitor software vulnerabilities, internal attackers, and security architecture. The decoy database collects data on SQL injections, credential hijacking, and privilege escalations used by attackers, which can be integrated into defense mechanisms and the security policy designed for the respective company.

- Malware: A malware honeypot simulates a software application or an Application Programming Interface (API) to generate malware attacks in a more controlled environment. This allows for the analysis of attack techniques and the development of new solutions to prevent malware or improve existing solutions that can address system vulnerabilities and potential threats.

- Spider Honeypots: The operation is similar to that of a spam honeypot,

  but in this case, the spider honeypot is designed to trap web crawlers or spiders by creating fake web pages and links.

It is also important to mention the concept of a honeynet [4][5]. A honeynet is the most complex type of honeypot, as it is a complete network composed of various systems that are ready to be attacked. Honeynets are high-interaction honeypots that act as an entire network specially designed to be attacked and collect more information about potential attackers. They use real equipment with real operating systems and running real applications. A honeynet can contain any existing network components, such as routers, switches, or any other element, allowing for the replication of any organization's network, no matter how complex it may be. This, combined with the use of real systems with real services and typical configurations, ensures that the risks and vulnerabilities encountered are exactly the same as those of the replicated organization's exposed systems.

2.3 Operation of a Honeypot

The operation of a honeypot involves several key stages, as its objective is to attract attackers and divert them from a real system. The main stages of this process are as follows:

- Honeypot Configuration: In this initial stage, the honeypot is configured to establish a controlled environment that simulates an attractive target with vulnerabilities to lure attackers. Realistic environments should be created with information and services that may be appealing to attackers. During this stage, the type of honeypot to be used should be defined, considering its level of interaction, and the desired characteristics to be simulated.

- Attractive Design: The honeypot must be designed in a way that is appealing and valuable to attackers while resembling the target of the network being protected. This involves deliberately implementing known vulnerabilities that are not obvious, so that attackers are enticed and actively engage with the honeypot, aiming to deceive them and reveal clues about the techniques they employ.

- Monitoring and Logging Activities: Once deployed in the controlled environment and operational, the honeypot starts recording all activities and events

that occur against it. Access attempts, types of attacks, methods used, and any information that attackers try to obtain are monitored. The generated logs are crucial for identifying attack patterns, detecting new threats, and designing security measures against them.

- Analysis and Response: The collected data is analyzed to understand the techniques used by the attackers. This analysis allows for improvements in security measures by identifying the exploited vulnerabilities.

To truly understand how a honeypot works, the stage of monitoring and logging is particularly important. Therefore, let's delve a little deeper into it. A honeypot primarily records events that occur in its surrounding environment. These events include port scans, vulnerability exploitation attempts, access attempts, service requests, and any actions performed by the attacker. These data are logged along with the attacker's IP address, attack type, attack duration, and the involved data. Network packets exchanged between the attacker and the honeypot are also captured to extract additional relevant data, such as the protocol or port used. In addition to event logging, the honeypot can also log the commands and actions executed by the attacker within the simulated environment, such as shell commands, file transfer requests, or database queries. If the honeypot simulates a file system, changes within it, such as file creation, modification, and deletion, can be logged as well.

2.4 Advantages

Honeypots offer several advantages [6], which include:

- Value of Data: One challenge faced by companies is extracting valuable data from the vast amount of collected information. Large companies generate massive amounts of data daily, including firewall logs, system logs, and intrusion detection alerts. This data overload makes it difficult to obtain valuable insights. Honeypots, on the other hand, collect a small amount of data but of significant value. This greatly reduces the noise level, going from collecting gigabytes of data daily to just a few megabytes or even less. Honeypots mainly capture scans, probes, and attacks, which are highly valuable data. A honeypot can provide the required information precisely in a fast and easily understandable format. Due to the limited data collected, it becomes easier to correlate and identify trends that most organizations would overlook.

- Resource Efficiency: Security mechanisms often face limitations and resource depletion. This occurs when a security mechanism can no longer function properly because its resources are overwhelmed due to high demand. A honeypot captures only the activities directed at itself, unlike a firewall or an IDS sensor that processes all network traffic. These mechanisms consume a significant amount of resources, and the speed and volume of traffic can be too high to

analyze every packet on the network. This may result in the omission of potential attacks. A honeypot avoids this issue by focusing only on the specific traffic directed at it. Another advantage of honeypots in terms of resources is that they do not require cutting-edge technology, high amounts of RAM, fast chip speeds, or large disk capacity for proper functioning.

- Simplicity: The advantage of simplicity in honeypots refers to the fact that complex algorithms, signature databases, and rule sets do not need to be designed, developed, and maintained. The procedure is the same even for research honeypots, which may be more complex. For all honeypots, all that is needed is to configure them, deploy them in the network, and wait for them to collect data. The simpler the operation, the more reliable it is, as increasing complexity can introduce potential errors.

- Return on Investment: Companies may start to believe that their investment is no longer profitable because threats are no longer being detected. Most security tools are costly investments that eventually become victims of their own success. By mitigating and eliminating a significant portion of existing threats, they may seem unprofitable. However, honeypots quickly and repeatedly demonstrate their value. Each time a honeypot is attacked, the existence of malicious activity is confirmed. By capturing unauthorized activity, honeypots can be used to justify not only their own value but also investments in other security resources. When a company's management perceives no threats, honeypots can effectively demonstrate the high level of risk that still exists.

2.5 Disadvantages

However, like any security tool, honeypots are not infallible and have their disadvantages [6]. Some of the drawbacks of honeypots are as follows:

- Narrow Field of View: A honeypot only detects activity explicitly directed at it. If an attacker enters the network and targets a system other than the honeypot, the honeypot will not be aware of it unless it is directly attacked. If it becomes apparent that a system is a honeypot, attackers may attempt to avoid it and infiltrate the network undetected. A honeypot acts as a microscope on the collected data, focusing on data value and the organization's security while excluding events happening outside its scope. This can be considered the biggest disadvantage of honeypots.

- Fingerprinting: Another disadvantage of honeypots, especially commercial ones, is fingerprinting. Fingerprinting refers to the traces left on the Internet that can reveal the nature of a system. In the context of honeypots, it refers to an attacker deducing that a system is a honeypot based on expected behaviors or common characteristics. For example, in a honeypot simulating an HTTP

server, if an attacker connects to this service and receives an HTML response with an error, they may deduce that it is a honeypot. An incorrectly implemented honeypot can even detect itself. Fingerprinting can be an even greater risk in research honeypots, as a system specifically designed for intelligence gathering can be in a precarious situation if detected. An attacker who has identified such a honeypot could provide it with misleading information, leading to erroneous conclusions. In most cases, companies do not want honeypots to be detected, although there are instances where a honeypot may identify itself in an attempt to deter the attacker.

- Risk: The third known disadvantage of honeypots is the risk they pose to the rest of the network. When we talk about risk in a honeypot, it means that an attacked honeypot represents a significant risk to the network in which it is deployed. Attackers can use the compromised honeypot as a stepping stone to target, infiltrate, or harm other systems and organizations. Some honeypots introduce very little risk, while others provide entire platforms from which new attacks can be launched. The simplicity of the honeypot is inversely proportional to the risk it poses. Risk levels vary between different honeypots, depending on their configuration and deployment.

Having examined these disadvantages, it can be concluded that honeypots cannot replace other security tools such as firewalls and intrusion detection systems (IDS). Honeypots should be used as a complementary tool alongside these other defense mechanisms.

Chapter 3

Criteria for Selecting the Right Honeypot

To choose the right honeypot, we need to consider the protocols that are relevant to the network we want to monitor. The typical protocols to consider in a public network are as follows:

- HTTP and HTTPS
- FTP and TFP
- DNS
- SMTP
- SSH
- Telnet
- SQL

Taking into account these protocols, I will now present a series of honeypots that support the monitoring of all or several of the mentioned protocols. But first, it is necessary to establish a taxonomy to explain the process of selecting the most suitable honeypot for the project.

3.1 Taxonomy for Classification

To classify the list of honeypots proposed for this project, we need a set of criteria that will be followed and compared against the objectives to be achieved. The classification criteria to consider in developing the honeypot taxonomy are as follows:

- Architecture

- Degree of interactivity
- Level of emulation
- Programming language used
- Objective

3.2 Architecture

When we talk about the architecture of a honeypot, we refer to how it is designed and built, including its components, their interaction, and how they are connected. It depends on the type of honeypot, its objectives, and the network in which it is being implemented. A typical average architecture includes the following components:

- Host: It is the hardware and operating system responsible for hosting the honeypot. It can be a physical or virtual machine, always configured in isolation to prevent attackers from accessing other network resources.
- Honeypot software: It is the specific software installed on the host to emulate the desired service or protocol to be protected. It can be customized or pre-configured and is used to attract attackers to interact with the honeypot.
- Honeypot network: It can consist of multiple honeypots interconnected to provide a more comprehensive view of attacker activity in a network. The honeypots in a network can be of different types and are configured to attract different types of attackers.
- Analysis tools: These are the tools used to analyze the collected data and generate reports on attacker activity. These tools may include network traffic analysis software, log file analysis, and malware analysis.
- Event Logging: This component is used to record and store the events of interaction between attackers and the honeypot. The log may include data such as attackers' IP addresses, attack techniques used, and the outcomes of the attack.
- Monitoring and Alerting: This component is used to monitor the honeypot and issue an alert if suspicious activity is detected or there are attempted attacks. It can be configured to notify security administrators in real-time or on a scheduled basis.

There are different types of honeypot architectures, some more specific and others more general. Here are the most common ones:

- Single System Honeypot: This is the simplest type of architecture. It consists of a single system running the honeypot software and is used to attract attackers. However, it can be easily evaded by attackers.
- High-Interaction Honeypot: This type is more complex and is used to emulate a real system. It offers a more comprehensive view of the techniques used by attackers as it allows full interaction with the emulated system. It requires more effort and maintenance than the previous type.
- Honeynet: It is a network of interconnected honeypots that aims to emulate a real network. It is used to attract attackers and provide a broader view of their attack techniques. It is very effective for security research but requires significant time and effort to implement and maintain.
- Low-Interaction Honeypot: It emulates a specific service or protocol. Unlike high-interaction honeypots, this type does not allow full interaction with the emulated system. This allows for simpler implementation and maintenance, but it is less effective in detecting more complex techniques.
- Cloud-based Honeypot: It runs in a cloud environment and is typically used to detect malicious activities in such environments. They are easy to deploy and have good scalability, but their interaction is limited by the cloud environment itself.

These are the most common types of architectures. There are other more specific architectures depending on the use case of the honeypot. Examples include modular or plugin-based architecture, web server-based architecture, host-based architecture, etc.

3.3 Degree of Interactivity

This refers to the level of interaction allowed with attackers. Honeypots can be classified as high-interaction, low-interaction, or somewhere in between.

- High-Interaction: These honeypots allow full interaction with the emulated system. Attackers can execute commands and perform activities similar to those in a real system. This provides greater visibility and understanding of attackers' tactics and techniques, but it also carries a higher risk as attackers could potentially use the honeypot as a point of entry to other systems in the network.
- Low-Interaction: These honeypots emulate only a specific part of the system, such as a protocol or service. They do not allow full interaction with the system. They are easier to implement and maintain and pose less risk to the network. The main drawback is that they may be less effective in detecting more advanced attacker tactics and techniques.

- Intermediate: These honeypots provide an intermediate level of interaction with attackers. They may emulate a part of the system, allowing more limited interaction than high-interaction honeypots. They are more effective than low-interaction honeypots but also more complicated to implement and maintain.

3.4 Emulation Level

The emulation level refers to the degree to which a honeypot emulates the service or protocol it aims to protect.

In a high-emulation honeypot, a real service is imitated with great precision. It can be more effective in attracting attackers as it provides a more realistic user experience and may lead attackers to believe they are interacting with a real system. The major disadvantage is that its configuration and maintenance costs are high due to the complexity of the simulation offered.

On the other hand, a low-emulation honeypot provides a less realistic simulation. Its main advantage is that it has lower maintenance and configuration costs. However, a significant drawback is that attackers may detect that they are interacting with a honeypot instead of a real system.

The choice of emulation level should depend on the objectives of using a honeypot. If the goal is simply to collect data on attackers' activity in a network, a low-emulation honeypot may be sufficient. However, if the goal is to analyze attackers' behavior, a high-emulation honeypot should be used.

3.5 Programming Language

Another criterion to consider in selecting the best honeypot for this project is the programming language in which it is developed. The choice of programming language for honeypot development depends on factors such as the type of honeypot, system resources, available network analysis tools, and the developer's preference and experience. Here are some common programming languages used in honeypot development:

- C/C++: It is used for its efficiency and low-level capabilities for system resource access.
- Python: It is popular due to its ease of use and a wide range of available libraries for network analysis tasks.
- Java: It is used for its portability and a broad selection of libraries for network-related tasks.

- Ruby: It is chosen for its ease of use and its ability to interact with databases.
- Bash/Shell: It is used for creating script-based honeypots as it is simple to use for task automation and file/directory manipulation.

Ultimately, the choice of programming language depends on the developer's experience. It would be more advantageous to choose languages that you have previous experience with.

3.6 Objectives

There are various objectives for which a honeypot can be used in a network. The following objectives can be considered:

- Threat detection: Honeypots are used to detect attacks and threats in real time. By emulating real-time applications and services, attackers can be attracted, and information about their tactics and tools can be collected.
- Information gathering: Honeypots are used to gather information about the techniques and tactics used by attackers. The collected data can be used to improve network security, identify attackers, and provide assistance in legal processes.
- Deception of attackers: Honeypots can be used to make attackers believe they have gained access to a valuable system. This can serve as a deterrent and reduce the chances of attackers targeting a real system.
- Research in the field of cybersecurity: Honeypots can be used in research to assess current security measures in the system, propose improvements to these measures, and develop new security tools.
- Training and education: Honeypots can be used to educate and train cybersecurity professionals in the techniques and tools used by attackers, enabling them to develop their skills in threat detection and response.

3.7 Open Source Honeypots

The following is a list of honeypots that I have considered to have some interesting features for the objectives sought in this project. These honeypots are all open-source, which means they can be modified to add additional functionality that may be of interest. All of these honeypots can be found in GitHub repositories.

Dionaea

Dionaea [7] is an open-source honeypot developed by DinoTools. It is a low-interaction honeypot that captures data on attacks and malware. It incorporates Python as its main programming language, but it also has modules programmed in C, offering a modular architecture. It uses libemu to detect shellcodes and libudns for DNS resolution, which is a non-blocking DNS resolution library. Additionally, it uses libev to be notified when it can use a socket for reading or writing. Dionaea offers network services via TCP/IP and TLS for both IPv4 and IPv6, and it can apply rate limits and accounting per connection to TCP and TLS connections if needed.

According to the official documentation, Dionaea's intention is to trap malware that exploits vulnerabilities exposed by services offered in a network, and its ultimate goal is to obtain a copy of the employed malware.

The software offers network services that may be exploitable. To minimize potential vulnerabilities, Dionaea can remove privileges and chroot. To execute actions that require elevated permissions after privileges have been removed, Dionaea creates a child process to perform those actions. This is a way to prevent obtaining root access to the system from an unprivileged user in a chroot environment.

Since it is software that offers network services, I/O (input and output) is crucial. All network I/O is handled within the main process in a non-blocking manner, using pipes so that when input data arrives, output data is written as long as the pipe is not full.

T-Pot

T-Pot [8] is a high-interaction honeypot based on a docker service emulation framework. T-Pot allows attackers to interact with simulated full operating systems, applications, and services to appear real, providing a higher level of realism.

Docker is used as the container platform for running the emulated systems and services, providing a secure and isolated environment that enables easier configuration and management of honeypots. T-Pot consists of various emulated services deployed in Docker, configured with known vulnerabilities to more easily attract attackers.

When an attacker interacts with the honeypot, all executed commands are recorded. It captures network packets, logs, and any other relevant information generated by the attacker. All stored data is subsequently analyzed to obtain information about

the techniques used by the attackers. T-Pot combines security detection and analysis services to monitor and analyze the generated traffic, helping to identify potential threats and attack patterns in real time.

ADBHoney

ADBHoney [9] is a low-interaction honeypot designed for Android Debug Bridge (ADB) over TCP/IP. Its goal is to trap any malware introduced by attackers targeting devices with the exposed port 5555. ADBHoney is developed in Python.

Android Debug Bridge (ADB) is a protocol designed for tracking real or emulated phones, TVs, and DVRs connected to a specific host. It implements various commands such as "adb push" or "adb pull" to assist developers in debugging and sending content to the device via a USB cable, with authentication and protection mechanisms. However, by executing a command like "adb tcpip <port>", the device is forced to expose its ADB services on port 5555. Once the services are exposed, one can connect to the device using the command "adb connect <ip>:<port>". TCP protocol lacks authentication mechanisms compared to USB, leaving the system vulnerable to attacks such as "adb shell <shell command>", which allows the developer to execute shell commands on the connected device, and "adb push <local file> <remote destination>", which enables the developer to upload binary files to the connected device. These two commands, along with the API, grant full control over the device as long as the port is exposed to the internet. The objective of ADBHoney is to capture malware being loaded through this exposed port on devices.

Ciscoasa

Ciscoasa [10] is a low-interaction honeypot used for the CISCO ASA component capable of detecting the CVE-2018-0101 vulnerability [11]. It is developed by the Massachusetts Institute of Technology (MIT) and released under a license for community use. It is primarily developed in JavaScript and Python.

CISCO ASA (Adaptive Security Appliance) are network security devices developed by the American company Cisco Systems. These devices provide a wide range of network services, from firewalls to VPNs and intrusion detection systems. They are commonly used in small and medium-sized enterprises to protect the internal network against external threats and enable communication with remote sites. Cisco ASA features include a firewall for controlling inbound and outbound network traffic, establishment of secure VPN connections for both site-to-site and client-to-site scenarios, intrusion prevention, application control, content filtering, high availability, and centralized management.

CVE-2018-0101, also known as Cisco ASA Remote Code Execution Vulnerability, is a vulnerability that allows an attacker to execute remote code or cause a denial-of-service (DoS) on a vulnerable device. This vulnerability is due to an error in the SSL (Secure Sockets Layer) or TLS (Transport Layer Security) packet processing on CISCO ASA devices with SSL/TLS VPN activated. Cisco discovered this vulnerability in January 2018 and provided software updates to address it. Therefore, this vulnerability is no longer exploitable, rendering the honeypot's purpose of detecting malware and malicious packets that trigger a DoS through this vulnerability obsolete.

Honeyd

Honeyd [12] is a low-interaction honeypot that offers a high level of emulation. Its goal is to detect and analyze intrusion attempts by attackers on a system. It consists of multiple hosts that can emulate various operating systems and services, creating a virtual network. It is programmed in Python.

According to the official documentation of this tool, Honeyd is a daemon that creates a network of virtual hosts, configurable to run arbitrary services and adaptable to appear as different operating systems. It allows a single host to claim multiple addresses, up to 65536 tested addresses, on a LAN to simulate a network. It discourages attackers by hiding real systems among virtual systems. It is possible to ping and trace route to the virtual machines. Any type of service can be simulated using a simple configuration file. It is also possible to proxy to another machine.

These brief insights into the functioning of Honeyd reveal that the tool can be used for both virtual network creation and general monitoring. The tool supports topology to create a virtual network, including routers, and can add latency and packet loss to add more realism to the simulation.

Kippo

Kippo [13] is a medium-interaction SSH honeypot designed to log brute-force attacks and all shell command interactions performed by the attacker during an SSH connection. It is developed in Python and inspired by, but not based on, Kojoney. It offers several interesting features, including:

- Fake filesystem with the ability to add and delete files. It resembles a Debian 5.0 installation.
- Ability to add fake contents for the attacker to view using commands like "cat".
- Minimal files like "/etc/passwd" are included.
- Session logs stored in a UML-compatible format.

- Saves files downloaded with wget for further analysis.
- SSH pretends to connect somewhere, but "exit" doesn't actually terminate the SSH session.

Cowrie

Cowrie [14] is a medium to high-interaction SSH and Telnet honeypot designed to capture brute-force attacks and the shell command interactions made by the attacker. When running in medium-interaction mode, it emulates a UNIX-like system, while in high-interaction mode, it acts as an SSH and Telnet proxy to observe the attacker's behavior towards another system.

In the default medium-interaction mode, Cowrie features include a fake filesystem based on Debian 5.0 where files can be added and deleted, saving files downloaded with wget, curl, or uploaded with SFTP for later analysis, and the ability to create fake files. The high-interaction mode, the SSH and Telnet proxy, allows running as a pure proxy or letting Cowrie manage a group of emulated QEMU servers. Both modes offer UML-compatible session logs, support for SCP and SFTP file uploads, SSH commands, logging of direct TCP connection attempts, JSON logs, and forwarding of SMTP connections to an SMTP honeypot.

Amun

Amun [15] is a low-interaction honeypot developed in Python and based on the concepts proposed by Nepenthes but with more sophisticated emulation and easier maintenance. It follows a modular architecture.

Amun focuses on emulating typical services such as web, email, and file servers. These services are configured to appear as real systems with visible vulnerabilities to make them more attractive to attackers. Lures are generated to attract attackers and increase interaction with the honeypot. Lures are emails, URLs, or files designed to look real. When interacting with these lures, malicious files and suspicious activities are captured for future analysis. It has an advanced emulation capability that can interpret and execute malicious code contained in the captured files, which is useful for conducting more thorough analysis. It is easy to configure and maintain.

Glastopf

Glastopf [16] is a web honeypot primarily developed in Python that emulates a vulnerable server hosting a variety of virtual machines and web pages with thousands of vulnerabilities.

Web application vulnerabilities, database vulnerabilities, and malicious scripts present an attack surface that can be exploited for spamming, turning websites into bots, and drive-by-download attacks. It employs vulnerability type emulation rather than vulnerability emulation. Once the vulnerability type is emulated, the tool can handle unknown attacks, but this can make the implementation more complex.

It features a modular architecture where new logging capabilities or attack type handlers can be added. The emulation of attack types is already implemented. Since attackers often use search engines, the honeypot uses keywords to attract attackers and extracts these keywords from incoming requests, automatically expanding its attack surface. Thus, the honeypot becomes more attractive to attackers with each new attack attempted against it.

Thug

Thug [17] is a low-interaction honeypot that mimics the behavior of a web browser capable of detecting and emulating malicious content. It is developed in Python and JavaScript and features a modular architecture.

It is implemented as a web browser emulator and runs in a controlled environment such as a virtual machine. Emulation techniques are used to interact with malicious websites and malware, allowing them to exploit vulnerabilities in the emulator. By behaving like a real web browser, it can detect techniques used by attackers, such as exploits, phishing, and malicious file downloads.

Thug is capable of executing JavaScript code found on webpages and downloaded files, allowing for the analysis of evasion techniques used by attackers. The tool extracts useful information such as server IP addresses, communication patterns, and obfuscation techniques.

Honeytrap

Honeytrap [18] is a low-interaction honeypot that aims to capture and analyze network traffic. It can be deployed on a single server as a standalone system or as a network of honeypots supporting complex architectures. Depending on the chosen deployment mode, it can listen on all ports to detect threats and gather information or listen on specific pre-defined ports and provide a specific response. Honeytrap supports various operating systems, with notable support for Linux, Windows, and macOS. This is beneficial for creating a virtual machine to deploy the honeypot. It features a modular architecture and is developed in C.

Honeytrap collects network packets sent to and from the honeypot, records source and destination IP addresses, ports, protocols used, and transmitted data. It logs commands and requests made by the attacker, such as port scanning, login attempts,

or database queries. It also logs files downloaded by the attacker from the honeypot for later analysis to determine their nature and potential associated threats.

It analyzes the behavioral patterns of attackers, determining the frequency of access attempts, the most targeted protocols and services, and the attack methods employed, providing a clearer understanding of attacker behavior. It also logs information about the attacker that can aid in tracking, with the IP address being particularly notable.

Conpot

Conpot [19] is an ICS (Industrial Control Systems) honeypot developed in Python. Its goal is to gather information about the methods used by attackers against industrial control software employed in electric, oil, gas, and other industries that rely on automation systems.

Conpot can be implemented on various operating systems. It is installed and configured to act as a service, simulating a real industrial control system. This involves utilizing typical protocols used in these systems, such as Modbus, DNP3, SNMP, and OPC. These systems have known vulnerabilities and inherently weak defense configurations, making them attractive targets for attackers.

When an attacker interacts with Conpot, all commands and actions they perform are logged. It can also interact with other honeypots and security tools to create more detailed reports and provide adaptability to different scenarios it may encounter.

Snare

Snare [20] is a low-interaction web honeypot focused on real-time event monitoring in Windows environments. It captures and logs events generated in real time. It is developed in C and C++.

Snare has an architecture based on an agent installed on a Windows system that collects events generated by the Windows system, such as login attempts or file changes. These events are sent over the network to a central server, where they are analyzed based on specific rules, allowing for the monitoring of all Windows systems that interact with it. This analysis helps detect anomalous patterns that may indicate intrusion attempts, security breaches, or malicious behavior in general.

Shockpot

Shockpot [21] is a web honeypot developed in Python, intended to find and detect attackers exploiting the remote code vulnerability in Bash known as CVE-2014-6271.

CVE-2014-6271 [11], also known as Shellshock, is a highly impactful and easily exploitable vulnerability that affects the Bash command interpreter on Unix systems. The Bash command interpreter interprets certain environment variables that contain user-defined functions as commands to be executed. Attackers can exploit this by injecting malicious code into these environment variables to remotely execute commands, escalate privileges, and gain access to sensitive data.

This vulnerability was patched shortly after its discovery, so the honeypot used to detect attackers exploiting it is no longer useful and is obsolete.

Honeycomb

Honeycomb [22] is an extension of Honeyd that inspects network traffic. It is recommended to use both tools together, Honeycomb and Honeyd, to simulate network services and attract attackers.

Honeycomb is developed in Python, has a high level of interaction, and is designed to generate authentication signatures for network intrusion detection systems (NIDS). It can also detect spam and worms. In conjunction with Honeyd, Honeycomb utilizes libcap (a C library) to detect the most relevant packets, avoiding duplication of efforts and allowing Honeycomb to focus on its functions. For example, with this combination, it is possible to determine the exact start and end of each connection.

Medusa

Medusa [23] is a low-interaction host-based honeypot used to gather information about attacks on Unix systems. It is developed in Python.

Medusa is deployed in a controlled environment, such as a virtual machine, that simulates a Unix operating system. When an attacker interacts with the honeypot, it logs the requests and techniques used. It can be configured in various ways to be adaptable to different scenarios and make it more enticing to attackers.

InetSim

InetSim [24] is a high-interaction honeypot developed in Python that simulates multiple services and aims to capture malicious activity that interacts with it.

It runs as a standalone application that emulates the following network services: HTTP, SMTP, DNS, and FTP. It uses emulation and redirection techniques to intercept attackers' requests and provide simulated responses. The captured information is then analyzed for early threat detection and understanding of attackers' techniques. InetSim is customizable and capable of adjusting its services to simulate specific scenarios and adapt to the needs of the situation.

3.8 Table for Honeypot Classification

Below, we will present all the previously discussed honeypots in a table 3.1 that allows for a combined, more direct, and visual selection criteria, making it easier to choose the most suitable honeypot.

3.9 Chosen Honeypot

Taking into account that the objective of the project is to collect data on a public network to demonstrate evidence of suspicious access to the network, it is reasonable to assume that we need a honeypot that provides a low level of interaction and preferably a high level of emulation, although this requirement is not as important in my opinion. Regarding programming languages, although I have experience in Python, I have more experience in C and C++, so it would be preferable to use either of the latter two programming languages.

After reviewing the honeypots presented in the table, I believe that the most suitable one, considering all the specified conditions, is Honeytrap, followed by Honeyid and, finally, Honeycomb.

All three meet the requirement of simulating network services to gather data about attackers. They all have a high level of emulation. However, only Honeyid and Honeytrap have a low level of interaction, while Honeycomb has a high level.

The honeypot that best fits the requirements of this project among the ones previously mentioned is Honeytrap.

However, Honeytrap needs to be modified to add functionality to record activity on the SSH, DNS, and HTTPS protocols since it only supports HTTP, FTP, and TFTP. As it is an open-source honeypot, new modules or plugins will be developed for the new protocols, opening the possibility of adding more protocols in the future. Once the design and development process is complete, it will be configured and deployed in a controlled environment to collect data.

It works from the code and documentation present in the Github repository https://github.com/armedpot/honeytrap/tree/master **??**. Based on this code I have added the modifications that can be seen in the chapter 5 in the sections 5.1, 5.2 and 5.3. These modifications add functionality for HTTPS, SSH, and DNS as well as support for SCP and SFTP downloads using an SSH connection. The repository where you can see the changes in the code in the following `https://github.com/SergioGarridoDeCastro/honeypot_TFG/tree/main`, also you can see these changes in Appendix II [8.2]

It is necessary to comment so far the original contribution with respect to the original code. The files htm_sshDownload.c and htm_dnsDetection.c as well as their respective header files (htm_sshDownload.h and htm_dnsDetection.h) have been added. These files are located in the src/modules directory.

Honeypot	**Architecture**	**Interaction Level**	**Emulation Level**	**Programming Language**	**Objectives**
Dionaea	Modular, plugin-based	High	High	Python	Capture malware and attack tools.
T-Pot	Various network services	High	High	Python	Collect information about attackers' tactics.
ADBHoney	Raspberry Pi-based	Low	High	Python	Collect malware targeting Android.
Ciscoasa	Virtual machine-based	Medium	High	JS and Python	Gather attack information on Cisco ASA devices.
Honeyd	Virtualized operating system	Low	High	C	Detect and analyze network service attacks.
Kippo	File system-based	High	Medium	Python	Gather attack information on SSH servers.
Cowrie	Virtualized operating system	Medium	Low	Python	Collect information and identify threats.
Amun	Modular architecture	Low	Medium	Python	Detect and analyze malware.
Glastopf	Web server-based	Low	Medium	Python	Detect and analyze web attacks.
Thug	Web browser-based	Low	High	Python	Analyze malware through web browsing simulation.
Honeytrap	Plugin-based, various network services	Low	High	C++	Collect and analyze malicious network traffic.
Conpot	ICS protocol emulation	High	High	Python	Emulate ICS and record attack attempts.
Snare	Windows system emulation	Medium	High	C++	Emulate Windows systems and capture malware.
Shockpot	Vulnerable web server emulation	High	High	Python	Record attack attempts on web servers.
Honeycomb	Plugin-based	High	High	Python	Simulate network services to monitor attackers.
Medusa	Host-based	High	Low	Python	Detect and gather attack information in UNIX systems.
InetSim	Network-based	Low	High	Python	Gather attack information on network services.

Table 3.1: Taxonomy to choose the right honeypot

The htm_httpDownload.c file has also been modified in the same directory to add the HTTPS functionality already mentioned.
In the src directory the files attack.c and attack.h have been modified to add a new struct to store the DNS queries. We have also modified ctrl.c, sock.h, sock.c, pcapmon.c, nfqmon.c, connectmon.c and dynsrv.c to correct several errors that have been appearing, mainly of obsolete code. The same has been done in /src/modules with the htm_ClamAV.c and htm_ClamAV.h files.
In the etc directory, the templates of the configuration files, honeytrap.conf.dist, and the port configuration file, port.conf.dist, have been modified in order to make the configuration easier once the application has been installed in the controlled environment where it will be deployed.

Chapter 4

Analysis and Design of Honeytrap Software

This chapter details the analysis and design phases of the software to be used in the project. When using existing software, the analysis methodology changes slightly from the conventional approach. The Unified Modeling Language (UML) is used for visual modeling and designing of software during the analysis and design phase.

Honeytrap is an open-source software, which means that the software can be freely modified and distributed. However, analyzing it becomes more complicated due to two reasons. The first reason is that it is existing software without documentation regarding existing use cases, their respective sequences, class diagrams, and entity-relationship diagrams that justify the domain model on which the software is based. There is also no information available about the design patterns being followed. We don't have the typical component, deployment, and sequence diagrams of the software design phase.

The second problem is that the software is developed in C. It is well-known that C is not an object-oriented language but a structured language. The traditional conception of software design teaches us that it is more common to work with an object-oriented approach.

4.1 Analysis I. Use Cases

Since Honeytrap is existing software, the detailed use cases may be imprecise, inaccurate, and somewhat detached from reality. They are assumptions based on observations of how the software functions.

The diagram 4.1 is a proposal based on the observed behavior of the software during testing, which depicts the most suitable use cases according to the original proposal, which is not provided in the technical documentation. Two actors that

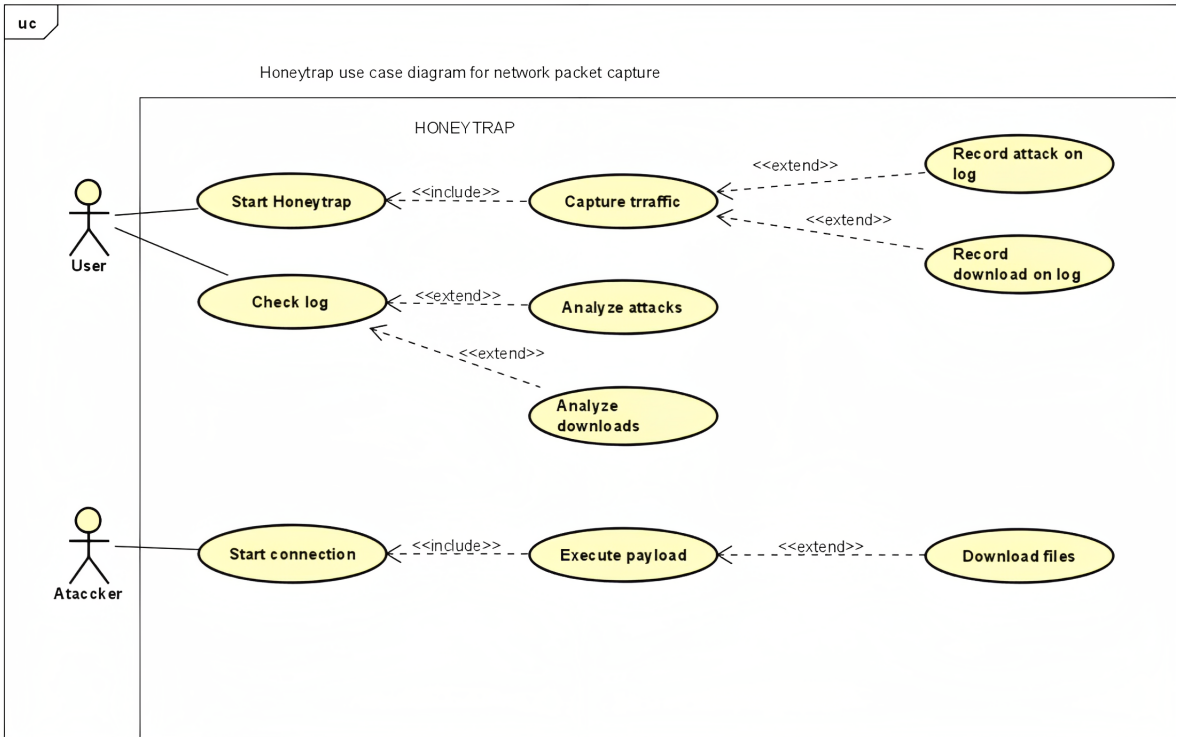


Figure 4.1: Use Case Diagram

interact with the Honeytrap system have been considered: User and Attacker. The first actor, User, interacts with the application to start it and check the logs. The "Start Honeytrap" use case extends to the "Traffic Capture" use case. Within the "Capture Traffic" use case, there are two included use cases: "Log Attacks" and "Log Downloads". The second actor, Attacker, interacts with the application to request a connection. If the connection is accepted, the attacker executes a payload. The "Execute Payload" use case extends to the "File Downloads" use case.

It is important to understand the concepts of "extend" and "include" in the use case diagram. The "«include»" relationship represents the inclusion of one use case within another. This means that the defined behavior of a use case is included within the main flow of another use case. In other words, an included use case is a functionality shared and reused by other use cases. The "«extend»" relationship represents a conditional or optional extension of a base use case. This means that the base use case can be extended with additional functionality without modifying the main flow, given certain conditions are met.

4.2 Analysis II. Domain Model

As mentioned before, Honeytrap is developed in C. Since C is a structured language and not object-oriented, the methodology for creating the domain model changes slightly. Instead of entities like in databases or objects in object-oriented languages,

the key concepts in the domain are the data structures or structs. These data structures can also have associated functions that act on the data structures to perform operations and manipulations within the domain.

4.2.1 Domain Objects. Data Structures

The diagram 4.2 proposes the domain objects as data structures with their own operations, attributes, and relationships, detailing the cardinalities. The data structures

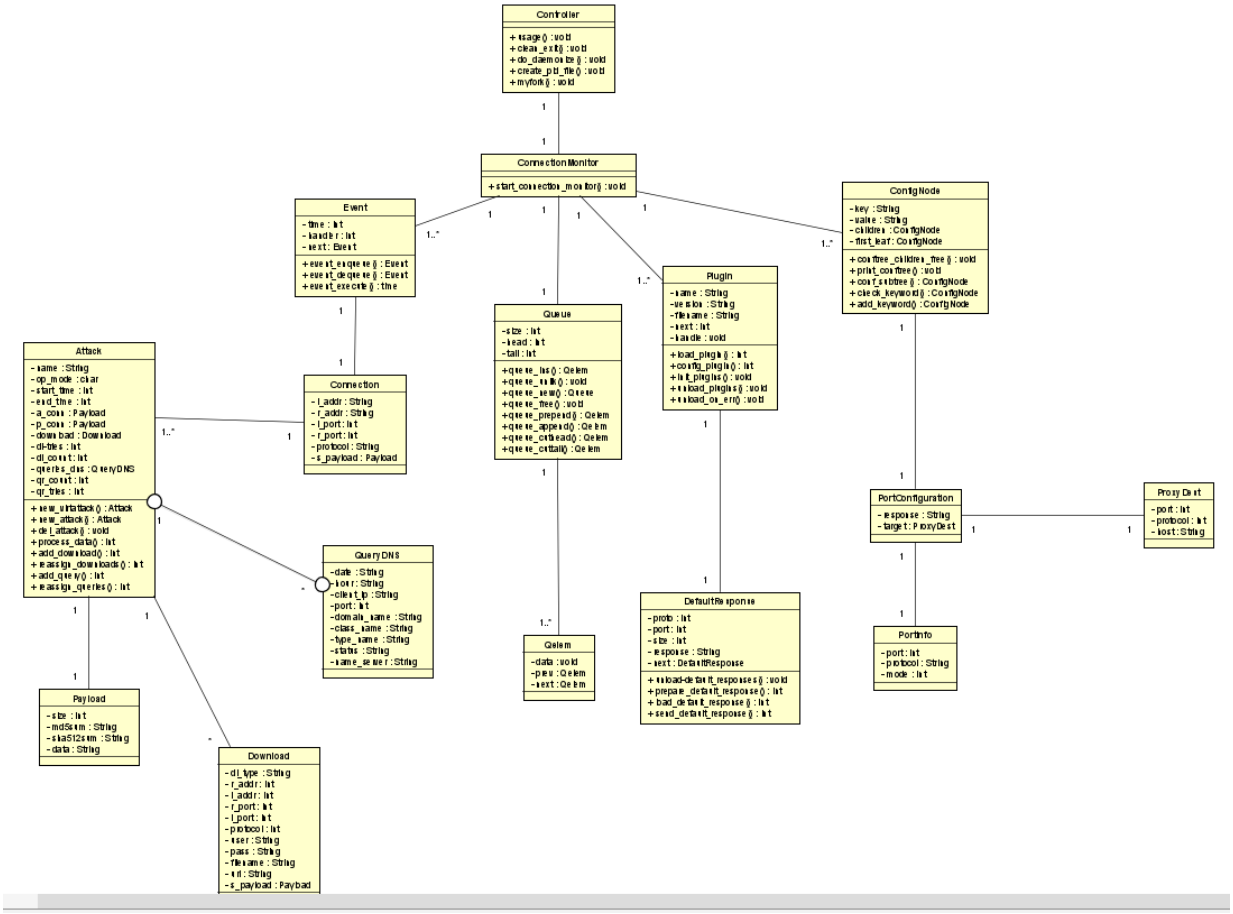


Figure 4.2: Data Structure Diagram

have been chosen by inspecting the existing structs in the header files (.h). Not all structures are included; only those considered as domain objects are included. There are data structures that do not represent any notable domain elements at a logical level but aid in coding and project development. According to the diagram, the following data structures have been proposed:

- Controller: Acts as the application controller. It is responsible for creating child processes, running the application as a daemon, generating a PID file, clearing the console output, and starting the application. This entity has no attributes as it is a controller of the application.

- ConnectionMonitor: Represents a connection monitor and is responsible for monitoring incoming connections. It has only one operation, start_connection _monitor, which initializes the connection monitor.

- Event: Represents an event. It includes operations related to event management (storing in a queue, removing from the queue, and executing) and attributes such as the timestamp, handler, and a pointer to the next event.

- Connection: Represents a connection with attributes that define a connection, such as local and remote addresses and ports, as well as the protocol and associated payload.

- Attack: Represents an attack carried out by an attacker on the honeypot. This is perhaps one of the most important entities in the entire domain due to the number of operations and attributes it has. The attributes of this data structure include the name, operation mode, start and end dates, addresses and ports, number of downloads and download attempts, as well as a vector that stores all the performed downloads. It stores the same data as the executed DNS queries, i.e., the query counter, attempt counter, and a vector that stores each executed DNS query. The operations defined for this data structure encompass all the actions an attack can perform: creating a new attack, creating a new virtual attack, deleting an attack from the vector, processing payload data, adding and reassigning both downloads and DNS queries to their respective vectors.

- Payload: Represents the payload used during an attack. It has no associated operations.

- Download: Represents a file download from the system. It has no associated operations and includes attributes related to a connection and the download itself (username, password, downloaded file name, and URI).

- QueryDNS: Represents a DNS request to the server. It includes attributes for all the fields of such requests, including the date, time, client IP address and port, domain name, class name, type name, status, and server name.

- Queue: Represents the queue that stores events. It has attributes such as size, the element at the head and tail of the queue. The defined operations are typical queue operations, such as creating a new queue, freeing its contents, inserting an element, removing an element from the queue, merging elements, and deleting the first or last element.

- Qelem: Represents each element of a queue. It only has attributes for the data of the element itself and two pointers that point to the previous and next elements.

- Plugin: Another important data structure in Honeytrap. Represents the structure of a plugin, which are modules that provide different and crucial functionality for the proper functioning of the application. Since each plugin has different functionality, this data structure defines the basic skeleton of a plugin, including attributes such as the name, version, executable file name, plugin handler, and the next plugin. It includes operations for loading, unloading, configuring, and initializing the plugin in the main program.

- DefaultResponse: Represents the default response executed by each plugin when an event occurs. It includes attributes such as the protocol used for the response, the port to send the response through, the network packet size, the response content, and a pointer to the next response. The operations defined are loading and unloading the default responses (the responses are stored in text or HTML files within the project), preparing these responses, and sending them.

- ConfigNode: Represents a configuration tree used to store configuration information. It includes attributes such as the key associated with the node, the value of the node, the child node, and the first leaf of the tree. The operations include typical tree operations, such as freeing a child node, configuring a subtree, printing the tree's contents, checking and adding new keys for a node.

- PortConfiguration: Represents the configuration of a port, including the response sent through that port and the proxy target to which the traffic is redirected.

- PortInfo: Represents information related to a port. The attributes include the port number, the listening protocol, and the mode in which the port is set.

- ProxyDest: Represents a proxy destination to which incoming traffic is redirected. The attributes include the port for traffic redirection, the protocol to be used, and the host address to which it will be redirected.

This is a proposal for the data structures and operations that can be performed. There are more operations in the code and more data structures, but they don't seem to affect the domain logic. Regarding the different modules, I have considered that by following a modular architecture, it was sufficient to explain the concept of a plugin in the domain, as their function within the domain is the same even though their functionality may change from one plugin to another. These data structures, along with the supported operations, are divided into source code files.

4.2.2 Relations

The first consideration in this model is to avoid the appearance of loops between the relationships of the elements within the domain model itself. There are more

relationships that could be apparent, but the diagram shows the simplest relationships following the principles of simplicity and transitivity, allowing entities to be related to each other transitively.

The controller is only related to the connection monitor, with only one connection monitor per controller. The events, queue, plugins, and configuration nodes are related to this connection monitor. There can only be one queue for each connection monitor, but there can be one or more plugins, events, and nodes related to a connection monitor. Each event represents a single connection. Within a connection, there can be one or multiple attacks. Each attack has a unique associated payload, but they can execute as many downloads and DNS queries as desired. The queue can have one or multiple elements. A node is associated with a unique port configuration, which in turn is associated with a unique PortInformation and a unique ProxyDest.

4.3 Design

This section will address the design of the application. After exploring the possible use cases, the data structures with their operations, and how they relate to each other, we will now discuss the approach used by Honeytrap in its design and the design patterns it is based on. As mentioned earlier, Honeytrap is an existing software, and we don't have technical documentation about these stages of software development. Therefore, the following assumptions are based on inspecting the code and understanding how the application works.

4.3.1 Approach

Honeytrap primarily adopts a modular approach. This can be inferred from the use of plugins in the application. A plugin allows for independent and loosely coupled extension of the system's functionality. Each plugin is developed and maintained independently, facilitating scalability, code reuse, and system evolution.

The modular approach is based on dividing a system into smaller, cohesive components or modules that interact with each other through a well-defined interface or core. In this case, plugins are used, where each plugin represents a module that encapsulates a specific and well-defined functionality. These plugins can be attached to or detached from the application's core without affecting its proper functioning. The plugins merely extend the system's functionality.

The modular approach provides several benefits over other approaches, such as ease of adding new functionalities without modifying existing code, the ability to reuse code and modules depending on the context, improved maintainability, and reduced coupling between different components of the system.

Based on this approach, we can consider the proposed core as the central part of the system. The core supports the basic functionality of the system, while the modules that add system functionality are the plugins. As mentioned earlier, the analysis defined the plugin data structure with its skeleton and basic functions, but couldn't provide details about the functionality of each plugin, as they follow a modular approach. The current system consists of 19 different modules with distinct functionalities. Here are the modules present:

- htm_baseDecode: This module is responsible for decoding strings that are encoded in base64.
- htm_ClamAV: This module detects malware using the ClamAV virus scanning engine.
- htm_cpuEmu: It implements CPU emulation. It provides functions to execute shell commands, create processes, and perform CPU emulation-related operations.
- htm_deUnicode: This module decodes strings that are encoded in Unicode.
- htm_dnsDetection: This module is designed to detect DNS queries. It provides functions to analyze network packets and determine if they contain DNS queries.
- htm_ftpDownload: It is designed to detect FTP connections and handles resource downloads via the FTP protocol. It provides functions to analyze FTP commands, establish FTP connections, and download resources via FTP.
- htm_httpDownload: This module is related to downloading resources via the HTTP and HTTPS protocols. It provides functions to analyze both HTTP and HTTPS URLs and perform corresponding download operations.
- htm_logAttacker: This module is responsible for logging attacks to a log file. It logs the attributes defined in the Attack and Payload structures of the domain model.
- htm_logJSON: This module logs attacks to a JSON file. Its functionality is very similar to the previous module, but it changes the file type and format of the generated log. The logged data remains the same.
- htm_magicPE: This module is responsible for detecting Portable Executable (PE) files.
- htm_SaveFile: As the name suggests, this module is responsible for saving data to a file. It provides functions to process configuration options and save the content of an attack object to a specified file.

- htm_SpamSum: This module implements the SpamSum algorithm to calculate a fingerprint of a message. It provides functions to calculate the fingerprint of an attack and perform operations related to the SpamSum algorithm.

- htm_sshDownload: This module is related to downloading resources via the SSH, SFTP, and SCP protocols. It provides functions to analyze SSH commands, establish SSH connections, and download resources via SCP and/or SFTP over an SSH connection.

- htm_submitMWserv: It sends information to a malware monitoring and analysis service. It provides functions to send attack data to the corresponding service.

- htm_submitNebula: It sends information to a Nebula service. It provides functions to send attack data to the Nebula service.

- htm_submitPostgres: It sends data to a PostgreSQL database. It provides functions to establish a connection to the database, process configuration options, and send attack data to the database.

- htm_tftpDownload: This module is related to downloading resources via the TFTP protocol. It provides functions to analyze TFTP commands, establish TFTP connections, and download resources via TFTP.

- htm_vncDownload: It is related to downloading resources via the VNC protocol. It provides functions to analyze VNC commands and download resources via SSH.

- htm_xmatch: This module provides the functionality of comparing patterns in received attack data. Its main purpose is to find matches between predefined patterns and attack data for decryption and decoding operations. If matches are found, the module performs additional processing of the attacks, such as analyzing decoded attacks and assigning possible downloads to the original attack.

4.3.2 Design Patterns

It is difficult to determine with certainty which design patterns have been followed during the design stage without documentation. However, by observing the project's code, certain common characteristics can be identified that suggest the following design patterns have been used in the code:

- Composite: According to the structure of the ConfigNode data, it has a "children" attribute that is a vector of child nodes. This suggests the use of the Composite pattern to represent a hierarchical tree structure, where a node can have child nodes.

The Composite pattern is used to represent objects, or in our case, data structures, in a hierarchical manner. It follows the principle of composing these objects in a tree-like structure and allows treating individual objects and object groups uniformly using a common interface. It consists of two main elements: the component and the composite. The component defines the common interface for all objects in the structure, while the composite is a concrete implementation of the component that can contain a collection of child components. This allows creating hierarchical structures of objects where both individual objects and groups of objects can be treated in the same way.

- Observer: The ConnectionMonitor structure is capable of monitoring connections, which may imply the use of the Observer design pattern, where the connection monitor acts as the subject, and the connections are the observers that receive updates when changes occur.
  The Observer pattern is used when there is a one-to-many relationship between objects, so that when one object changes its state, all dependent objects are notified and updated automatically. The Observer pattern is based on the principle of separating the update logic of observer objects from the logic of the observed objects. In this pattern, there are two types of objects: the subject or observable and the observers. The subject maintains a list of registered observers and provides methods for observers to register or unregister themselves. When the subject changes its state, it automatically notifies all observers by calling an update method defined in the observer's interface.

- Command: The Attack data structure is used to represent specific commands or actions, suggesting the use of the Command pattern, where commands are encapsulated and executed flexibly.
  The Command pattern is used to encapsulate a request as an object, allowing clients to be parameterized with different requests, queue or log requests, and support undoable operations. This pattern is based on the principle of encapsulating a request as a separate object, which allows treating requests as first-class entities. It has four main components: the command, the receiver, the invoker, and the client. The command encapsulates a request as a concrete object and has a reference to the receiver, which is responsible for carrying out the action associated with the request. The invoker takes the command and executes it when required. The client creates the command and sets its receiver.

It should be noted that in our case, the application is programmed in C, which is not an object-oriented language. Therefore, the objects mentioned in the design pattern definitions actually refer to data structures, which in our case are defined in the domain model. With that clarification, it can be said that the Composite pattern can be implemented in C using a structure that acts as the base component and pointers to other components or subcomponents of the same data type. The

Observer pattern can be implemented in C using a structure that represents the observed subject and a list of function pointers (observers) to be called when an event occurs. The Command pattern can be implemented in C using a command structure that encapsulates a request and a function pointer to be executed when the command is invoked.

Although there is not as clear evidence in the code as for the previous patterns, it can also be considered that the following design patterns have been used due to the system's modular approach.

- Strategy: It is a pattern used when different interchangeable algorithms and strategies are desired. In the case of a modular application with plugins, it is recommended to use this pattern to implement a strategy for selecting the appropriate plugin based on needs.
- Factory: This pattern is used for object creation. In the context of an application that uses plugins, it is interesting to consider implementing a factory to dynamically create instances of plugins, allowing the incorporation and loading of new plugins at runtime.

Chapter 5

Configuration of the Chosen Honeypot

After explaining how the honeypot software is designed, this chapter will cover the configuration used during the application's execution. It will also discuss the modifications made to the application's code to add new functionality with new plugins.

As mentioned in Section 3.7 of Chapter 3, Honeytrap is a low-interaction honeypot designed to capture and analyze network traffic. It follows a modular approach with a set of plugins that provide different functionalities. According to the original application, it offered support for monitoring TFTP, HTTP, VNC, and FTP traffic, as well as features for algorithms, log file handling, data forwarding to different services, pattern matching, emulation, and decoding. Based on the protocols deemed interesting to monitor and comparing them with the supported protocols in Honeytrap, it was decided to modify an existing plugin to add support for the HTTPS protocol and create new plugins to support DNS, SSH, SCP, and SFTP protocols.

5.1 Plugin providing support for HTTP

This module only provided functions to analyze HTTP URLs and perform resource downloads through the HTTP protocol. However, the HTTP protocol is not secure because data is transmitted in plain text. This means that anyone intercepting the communication can easily read and understand the exchanged information. This poses a problem when exchanging sensitive data such as login information or personal data. To address this, an updated version of the protocol was introduced to add an additional layer of security called SSL/TLS (Secure Sockets Layer/Transport Layer Security) to encrypt data during transfer. This ensures that the data is only comprehensible to the server and client participating in the communication, making it more difficult for third parties to intercept and read the data.

For this reason, it was considered appropriate to incorporate functionality for the HTTPS protocol as well. It all starts with a function called int is_https(char *request) 8.3.1 that can detect if the URL's header is *"https://"* or *"http://"*.

In the module, there is a function called ***int cmd_parse_for_http_url(Attack *attack);*** 8.3.2 that searches for URLs in an attack string. If found, it initiates a download using the 'wget' program and executes a command to download the file. To add functionality for HTTPS, this code block has been added within the function.

As you can see, it is a small modification that allows us to add functionality for HTTPS and further expand the range of capabilities offered by Honeytrap. It was decided to modify this module instead of creating a new one due to the close relationship between the two protocols, as one is the evolution of the other, and this way we also avoid excessive code repetition.

5.2 Plugin providing support for DNS

Although the code related to the handling of a plugin by the main core of the application will be repeated, it is necessary to create a new module to support the DNS protocol, as it is a completely different protocol.

DNS translates domain names into IP addresses through a series of queries and responses between DNS servers to enable effective communication between computers on the Internet. When working with a honeypot, the received communications are incoming, so the logic tells us that we should focus mainly on DNS requests. That's why code has been implemented only to handle a DNS request.

We will omit all the code related to handling a plugin to focus exclusively on the functionality related to detecting and handling incoming DNS requests. The function int is_dns_query(char* packet) 8.4.1 receives a packet and checks if it has the minimum size that a DNS packet should have. If it meets the size requirement, it determines whether it is a response or a request by checking the value of the query/response bit.

The function int cmd_parse_for_dns_query(Attack *attack, struct dns_query *query) 8.4.2 is responsible for parsing the content of the DNS packet to find the data for the fields specific to a DNS request and add them to the log. According to the typical format of a DNS request, which is as follows: <dd-mmm-YYYY HH:MM:SS.uuu> <client IP><port> query: <query_Domain name> <class name> <type name> <- or +>[SETDC] <(name server IP)>, the data sought in the string are the date, time, client IP and ports, domain name, class, type, and server.

5.3 Plugin providing support for SSH, SCP, and SFTP

In this case, it is also important to create a new plugin as it will provide functionality for handling connections using the SSH protocol and SCP and SFTP downloads. SSH is a network protocol that allows secure and encrypted communication between two devices. It is primarily used for secure remote access and administration of systems. It provides authentication, confidentiality, and integrity of transmitted data. SCP and SFTP are two protocols for remote file transfer. SCP allows secure file transfer between a client and a remote server. It uses SSH for authentication and encryption. SFTP is an advanced version of SCP that supports secure and encrypted file transfer, as well as synchronization and the ability to set file permissions and attributes. It also uses SSH infrastructure to secure transmissions, ensuring the confidentiality and integrity of transferred data.

The function int cmd_parse_for_ssh (Attack * attack) 8.5.1 is responsible for parsing the attack string passed to it to find evidence of an SSH command. If found, it calls the ‘get_sshcmd‘ function.

The function int get_sshcmd(char *attack_string, uint32_t string_size, Attack *attack) 8.5.2 continues parsing the attack string once evidence of an SSH command is found. It parses the string to save each element of an SSH command according to the standard format: ***ssh [options] [user@]host [command]***

The function int get_ssh_resource (const char * user , const char * host , const char * remote_path , const char * local_path , 2 Attack * attack , const char * conn_type , const char * filename) 8.5.3 is responsible for downloading the requested resource based on the arguments passed in the SSH command, once the SSH command is identified.

The function int get_ssh_ resource_by_sftp (const char * user , const char * host , const char * remote_path , const char * local_path , Attack * attack , ssh_session ssh) 8.5.4 handles resource downloads using the SFTP protocol, by establishing an SFTP connection using an existing SSH session. It then opens a remote file in read mode, creates and opens a local file in binary write mode. It reads the contents of the remote file in fragments and writes them to the local file. Finally, it closes both files, disconnects the SSH session and logs the connection in an Attack object. Returns 0 on success and -1 on error.

The function int get_ssh_ resource_by_scp (const char * user , const char * host , const char * remote_path , const char * local_path , Attack * attack , ssh_session ssh) 8.5.5 handles resource downloads using the SCP protocol. In this function a download of a remote file is performed via SSH using the SCP protocol. Opens an

SCP session, initializes the session, opens the remote file in read mode, creates and opens the local file in binary write mode, reads the contents of the remote file and writes it to the local file in 1024-byte chunks, closes both files, disconnects the SSH session and adds a connection record to the Attack object. Returns 0 on success and -1 on error.

5.4 Configuration File

Once these modifications have been developed, the program is configured, compiled, and installed using the following command, which allows us to configure the application with the logattacker, logjson, clamAV, xmatch, cpuEmu, spamsum, and magicPE plugins. The chosen method for network traffic monitoring is nfq or packet capture.

```
./configure  --with-stream-mon=nfq  --with-logattacker --with
-logjson --with-clamAV --with-xmatch --with-cpuEmu --with-spamsum
--with-magicpe &&  make &&  sudo make install
```

Honeytrap has three monitoring modes that can be configured in the application, and the functionality changes based on the installed method. There are preprocessing options in the code for each monitoring method. The available modes are:

- NFQ: Uses the libnetfilter_queue library for monitoring. It is compatible only with Linux. The libnetfilter_queue library allows capturing and manipulating network packets through the netfilter queuing interface. It provides a high level of control and flexibility for connection monitoring.
- IPQ: This option uses netfilter/iptables' ip_queue for monitoring. It is also compatible only with Linux. Netfilter/iptables is a packet filtering infrastructure built into the Linux kernel. Using ip_queue allows capturing and examining packets using iptables rules before filtering decisions are made.
- PCAP: Uses a PCAP-based sniffer for monitoring. PCAP is a library used to capture and analyze network packets. This monitoring method is independent of the operating system and can be used on different platforms.

However, before running the program, it is necessary to modify its configuration file to enable proper monitoring. The initial configuration to be used during execution is as follows:

```
1 // log to this file
2 logfile         = "/usr/local/etc/honeytrap/honeytrap.log"
3
4 // store process ID in this file
5 pidfile         = "/var/run/honeytrap.pid"
```

```
6
7  /* where to look for default responses
8   * these are sent for connections handled in "normal mode" */
9  response_dir    = "/usr/local/etc/honeytrap/responses"
10
11 // replace rfc1918 IP addresses with attacking IP address
12 replace_private_ips = "no"
13
14 // bind dynamic servers to a specific address
15 //bind_address = "127.0.0.1"
16
17 /* put network interface into promiscuous mode
18  * (only available when compiled with --with-stream-mon=pcap)
      */
19 //promisc = "on"
20
21 /* the user and group under which honeytrap should run
22  * should be set to non-root */
23 user    = "user_Sergio"
24 group   = "group_tfg"
25
26 // do not read more than 20 MB - used to prevent DoS attacks
27 read_limit = "20971520"
28
29
30 /* ----- plugin stuff below ----- */
31
32 /* where to look for plugins
33    needs to be set before loading plugins */
34 plugin_dir      = "/usr/local/etc/honeytrap/plugins"
35
36
37 // include a plugin via plugin-[ModuleName] = ""
38
39 plugin-magicPE = ""
40 plugin-ftpDownload = ""
41 plugin-tftpDownload = ""
42 plugin-b64Decode = ""
43 plugin-deUnicode = ""
44 plugin-vncDownload = ""
45 plugin-dnsDetection = ""
46 plugin-sshDownload = ""
47
48
49 // store attacks on disk
50 plugin-SaveFile = {
51         attacks_dir     = "/opt/honeytrap/var/honeytrap/
      attacks"
52         downloads_dir   = "/opt/honeytrap/var/honeytrap/
      downloads"
53 }
54
```

```
55
56 // plugin for shellcode detection and emulation
57
58 plugin-cpuEmu = {
59         execute_shellcode = "no"
60         createprocess_cmd = "/bin/sh -c \"cd /opt/honeytrap-
       libemu/.wine/drive_c/windows/system32; WINEPREFIX='/opt/
       honeytrap-libemu/.wine/' WINEDEBUG='-all' wine 'c:\\windows
       \\system32\\cmd_orig.exe'\""
61 }
62
63
64
65
66 // scan downloaded samples with ClamAV engine
67 /*
68 plugin-ClamAV = {
69         temp_dir        = "/tmp"
70         clamdb_path     = "/var/lib/clamav"
71 }
72 */
73
74
75 // calculate locality sensitive hashes
76
77 plugin-SpamSum = {
78         md5sum_sigfile  = "/opt/honeytrap/md5sum.sigs"
79         spamsum_sigfile = "/opt/honeytrap/spamsum.sigs"
80 }
81
82
83
84 // store attacks in PostgeSQL database
85 /*
86 plugin-SavePostgres = {
87         db_host = "localhost"
88         db_name = "some_db"
89         db_user = "some_user"
90         db_pass = "some_pass"
91 //      db_port = "some_port"   // defaults to 5432/tcp if not
       set
92 }
93 */
94
95
96 // invoke an external program (f.e. wget) to download files
       via http
97
98 plugin-httpDownload = {
99         http_program = "/usr/bin/wget"
100        http_options = "-q -t1 -T1 -O-"
101 }
```

```
102
103
104
105 // submit downloaded malware samples to the mwcollect alliance
106 /*
107 plugin-submitMWserv = {
108         mwserv_url      = "https://submission-url/"
109         guid            = "your-guid"
110         maintainer      = "your-maintainer"
111         secret          = "your-secret"
112         timeout         = "120"
113 }
114 */
115
116 // log attacker connection information to a separate file, one
     entry per line
117
118 plugin-logAttacker = {
119         logfile = "/opt/honeytrap/attackers.log"
120 }
121
122
123 // log attack details in JSON format
124
125 plugin-logJSON = {
126         logfile = "/opt/honeytrap/attackers.json"
127 }
128 /* ----- port mode configuration below ----- */
129
130 // default port configuration (ignore, normal or mirror)
131 //   ignore: just ignore connection attempts
132 //   normal: send a default response
133 //   mirror: mirror connections back to the initiator (use
    with caution!)
134 portconf_default = "normal"
135
136 // explicit port configuration
137
138 portconf = {
139         ignore = {
140                 protocol        = "tcp"
141                 port            = ["25", "1433"]
142         }
143         normal = {
144                 protocol        = ["tcp", "udp"]
145                 port            = ["53"]
146         }
147         mirror = {
148                 protocol        = ["tcp", "udp"]
149                 port            = ["23", "80", "8080", "69",
    "22", "5353", "443", "8443"]
150         }
```

```
151         proxy = {
152                 proxy-http = {
153                         protocol        = []
154                         port            = []
155
156                         target_host     = ""
157                         target_protocol = ""
158                         target_port     = ""
159                 }
160                 proxy-tftp = {
161                         protocol        = []
162                         port            = []
163
164                       target_host     = ""
165                         target_protocol = ""
166                         target_port     = ""
167                 }
168
169                 proxy-ssh = {
170                         protocol        = []
171                         port            = []
172
173                         target_host = ""
174                         target_protocol = ""
175                         target_port = ""
176                 }
177
178                 proxy-dns = {
179                         protocol        = []
180                         port            = []
181
182                         target_host = ""
183                         target_protocol = ""
184                         target_port = ""
185                 }
186
187                 proxy-https = {
188                         protocol        = []
189                         port            = []
190
191                         target_host = ""
192                         target_protocol = ""
193                         target_port = ""
194                 }
195         }
196 }
```

In this configuration file, the directory where the main log of Honeytrap activity will be stored is defined. It also specifies the file that stores the PIDs of the parent process and child processes, and the directory where the responses are stored when

there are connections on the ports configured in normal mode. The user and group with permissions to execute are set. It is established that promiscuous mode will not be used, and the IP addresses of attackers will not be replaced with fake IP addresses because our goal is traffic analysis on the network. The dynamic servers are not bound to a specific address (127.0.0.1), and the read limit is set to 20 MB to avoid possible DoS (Denial of Service) attacks.

It is configured to enable the plugins mentioned above, while the rest of the plugins are commented out. The directory where the plugin executables are located for loading them is /usr/local/etc/honeytrap/plugins. The directories where intercepted attacks and downloads are stored, the commands executed in CPU emulation, and the HTTP downloads are defined. There are also files where hashes and log data for intercepted data are stored.

Finally, an explicit port configuration is provided. Port 22 is ignored, port 53 is set to normal mode, and no proxy is configured for any port. The remaining ports are set to mirror mode, which includes ports: 23, 80, 8080, 69, 22, 5353, 443, 8443.

Mirror mode implies that connections are sent back to the attacker. In normal mode, default responses are sent to the attacker, and ignore mode simply ignores connection attempts.

5.5 Configuration of Vulnerable Services

After analyzing the collected data in the initial phase, it was concluded that the honeypot was not attractive enough to attackers. Therefore, the decision was made to install and configure vulnerable services for each port that the honeypot listens on:

- Port 20 and 21 (FTP): An anonymous and weakly credentialed FTP server has been installed and configured to allow unauthorized access.

```
1         # Install an FTP server
2         sudo apt-get install vsftpd
3
4         # Configure the FTP server to be vulnerable
5         sudo vi /etc/vsftpd.conf
6
7         # Add configuration settings that introduce
    vulnerabilities allowing anonymous access and file
    writing.
8         anonymous_enable=YES
9         write_enable=YES
10        # ...
11
```

```
12          # Restart the FTP server to apply the changes
13          sudo service vsftpd restart
14
15
```

- Port 22 (SSH): The SSH service is configured to allow less secure access or use a vulnerable version of OpenSSH.

```
1           # Configure less secure SSH access
2           sudo vi /etc/ssh/sshd_config
3
4           # Allow root access and disable public key
    authentication
5           PermitRootLogin yes
6           PubkeyAuthentication no
7
8           # Restart the SSH service to apply the changes
9           sudo service ssh restart
10
11
```

- Port 23 (Telnet): A Telnet server with vulnerabilities has been configured.

```
1           # Install the Telnet server
2           sudo apt-get install telnetd
3
4           # Configure the Telnet server to be vulnerable
5           sudo vi /etc/inetd.conf
6
7           # Comment out the line for regular Telnet and
    uncomment the line for vulnerable Telnet
8           telnet  stream  tcp nowait  telnetd /usr/sbin/tcpd
     in.telnetd # Comment out
9           telnet  stream  tcp nowait  root     /bin/echo
    echo vulnerable; /bin/bash # Uncomment
10
11          # Restart the inetd service to apply the changes
12          sudo service inetutils-inetd restart
13
14
```

- Port 53 (DNS): BIND, an open-source DNS server, has been installed with a vulnerable version. The following steps were taken for the installation.

```
1           # Install BIND
```

```
2         sudo apt-get install bind9
3
4         # Configure BIND to be vulnerable
5         sudo  /etc/bind/named.conf.options
6
7         # Add configurations that introduce vulnerabilities
    , allowing recursive queries from any IP address.
8         options {
9             recursion yes;
10            allow-recursion { any; };
11            // ...
12        }
13
14        # Restart BIND to apply the changes
15        sudo service bind9 restart
16
```

- Port 69 (TFTP): A vulnerable TFTP server has been installed and configured.

```
1         # Install a TFTP server
2         sudo apt-get install tftpd-hpa
3
4         # Configure the TFTP server to be vulnerable
5         sudo nano /etc/default/tftpd-hpa
6
7         # Modify the configuration to introduce
    vulnerabilities allowing file writing.
8         TFTP_OPTIONS="--secure --create"
9
10        # Restart the TFTP server to apply the changes
11        sudo service tftpd-hpa restart
12
13
```

- Port 80 (HTTP): Apache has been installed and configured with a vulnerable version.

```
1         # Install Apache
2         sudo apt-get install apache2
3
4         # Configure Apache
5
6  to be vulnerable
7         sudo vi /etc/apache2/apache2.conf
8
9         # Add configurations that introduce vulnerabilities
    , allowing script execution in all directories.
10        <Directory /var/www/html>
```

```
            Options +ExecCGI
            AddHandler cgi-script .cgi .pl
        </Directory>

        # Restart Apache to apply the changes
        sudo service apache2 restart


```

- Port 8080 (Alternative HTTP): An alternative web server, Apache Tomcat, has been installed and configured with a vulnerable version.

```
        # Install Apache Tomcat
        sudo apt-get install tomcat9

        # Configure Apache Tomcat to be vulnerable
        sudo nano /etc/tomcat9/server.xml

        # Modify the configuration to introduce
    vulnerabilities, allowing access without authentication.
        <!-- Define a new HTTP connector on port 8080 -->
        <Connector protocol="HTTP/1.1" port="8080"
    connectionTimeout="20000" redirectPort="8443" />

        # Restart Apache Tomcat to apply the changes
        sudo service tomcat9 restart


```

- Ports 443 and 8443 (HTTPS): The same Apache server as before has been configured, but this time with vulnerabilities for HTTPS.

```
        # Configure Apache to be vulnerable on ports 8443
    and 443
        sudo vi /etc/apache2/sites-available/default-ssl.
    conf

        # Add a VirtualHost with SSL
        <VirtualHost *:8443>
            SSLEngine on
            SSLCertificateFile /home/ubuntu/vulnerable.crt
            SSLCertificateKeyFile /home/ubuntu/vulnerable.
    key
            # ...
        </VirtualHost>

        # Create vulnerable.key and vulnerable.crt
```

```
13
14          # Enable the VirtualHost and SSL
15          sudo a2ensite default-ssl
16          sudo a2enmod ssl
17
18          # Restart Apache to apply the changes
19          sudo service apache2 restart
20
21
```

Chapter 6

Deployment in the Public Network

In this chapter, we will explain the deployment and execution process in a controlled environment.
Since Honeytrap requires an Ubuntu operating system or similar Linux/UNIX distributions and it is a security tool, it is advisable to use a virtualized environment to avoid potential unwanted intrusions on a personal computer or server. It is also worth mentioning that the execution will take place for an extended period, so it is not recommended to set up a virtualized environment on a personal computer. Therefore, the most recommended option is to use a virtualized environment on a cloud hosting platform.

After comparing various hosting service providers, I have decided to use AWS (Amazon Web Services). There are three main reasons behind this decision. The first and primary reason is the cost of hosting a project of this size. Since it is an application deployed on a virtual machine that generates log files, significant resources are not required. AWS is cheaper than its competitors and offers a free tier for the first year, subject to certain usage conditions. The second reason is that AWS provides a wide variety of tools that can complement the virtualized environment. The third reason, which is related to the previous one, is the abundance of technical documentation and forums available to address any potential issues.

The most relevant AWS tool for creating a virtualized environment to host the honeypot is EC2 (Amazon Elastic Compute Cloud). An EC2 instance has been created to host the honeypot with the following specifications:

- Operating system based on Linux/UNIX. The chosen distribution is Ubuntu, as it is the most stable distribution with a larger community to provide support and documentation.
- It is an instance of type t2.micro.

- 1 virtual CPU.
- 8 GB of internal storage.
- 1 GB of RAM.
- HVM virtualization type.
- Custom high-frequency Intel Xeon processors with adjustable scale and Intel AVX-512 instructions.
- Public IPv4 address: 35.180.174.253
- Public IPv4 DNS: ec2-35-180-174-253.eu-west-3.compute.amazonaws.com
- Private IP address: 172.31.43.196.
- Private IP DNS name (IPv4 only): ip-172-31-43-196.eu-west-3.compute.internal
- Root device name: /dev/sda1.
- Root device type: EBS.
- Existence of a subnet with 4089 available IPv4 addresses, automatic assignment of the public IPv4 address. Its subnet ID is subnet-0a58a0e1a7abba58c, and its subnet ARN is arn:aws:ec2:eu-west-3:670690652774:subnet/subnet-0a58a0e1a7abba58c.
- Existence of a VPC (Virtual Private Cloud) with ID vpc-0831080668b25f3db. VPC is an AWS service that allows the creation and configuration of an isolated virtual network in the cloud. Its IPv4 CIDR is 172.31.0.0/16. DNS hostnames and DNS resolution are enabled.

The following inbound rules have been configured in the security group as detailed in Table 6.1 and 6.2. Additionally, an outbound rule has been configured as follows:

Security Group Rule ID	IP Version	Type	Protocol	Port Range	Source
sgr-0840162eb193c2090	IPv4	All traffic	All	All	0.0.0.0/0

Table 6.1: Regla de salida

After setting up these rules, we connect to the instance using the following SSH command:

```
ssh -i "mvTFG.pem" ubuntu@ec2-35-180-174-253.eu-west-3.compute.
amazonaws.com
```

Security Group Rule ID	IP Version	Type	Protocol	Port Range	Source
sgr-0a6fab604b67f22c5	IPv4	Custom UDP	UDP	69	0.0.0.0/0
sgr-00632bb4359affbd2	IPv4	SSH	TCP	22	0.0.0.0/0
sgr-047b0868e7b3d5038	IPv4	Custom TCP	TCP	8443	0.0.0.0/0
sgr-06161ce2ddc71ebcd	IPv4	HTTPS	TCP	443	0.0.0.0/0
sgr-025de514a81883d20	IPv4	Custom TCP	TCP	2222	0.0.0.0/0
sgr-01168845332dc172f	IPv4	Custom TCP	TCP	8080	0.0.0.0/0
sgr-0a232fe4220fe748d	IPv4	HTTP	TCP	80	0.0.0.0/0
sgr-068b6ef982e5e9ae0	IPv4	Custom UDP	UDP	5353	0.0.0.0/0
sgr-0c8d28605daa31113	IPv4	SMTP	TCP	25	0.0.0.0/0
sgr-0bdd3e75a70ca237a	IPv4	MSSQL	TCP	1433	0.0.0.0/0
sgr-00eb1299f3aaa6942	IPv4	DNS (UDP)	UDP	53	0.0.0.0/0
sgr-0f9e25c607d09ecf9	IPv4	Custom TCP	TCP	23	0.0.0.0/0

Table 6.2: Table of inbound rules in the instance

First, the project needs to be transferred as a compressed zip file using scp:

```
scp -i "mvTFG.pem" honeytrap.zip ubuntu@ec2-35-180-174-253.eu-west
-3.compute.amazonaws.com:honeytrap.zip
```

After configuring, compiling, and installing as mentioned in section 5.4 of chapter 5, Honeytrap is then executed using the following command:

```
sudo /usr/local/sbin/honeytrap i eth0 -t 3
-C /usr/local/etc/honeytrap/honeytrap.conf
```

To this command, superuser permissions are granted so that it can create files and write to files that have special permissions either by themselves or in the directories where they are located. /usr/local/sbin/honeytrap is the path to the Honeytrap executable, eth is the interface from which the traffic should be listened to, -t 3 indicates the level of detail in the log, which is the normal or default level (e.g., debug is -t 6), and -C /usr/local/etc/honeytrap/honeytrap.conf is the configuration file for Honeytrap. Once it is executed and if there are no errors, it is daemonized and the instance can be used for other functions, although it is not entirely recommended as detecting SSH traffic can clutter the honeypot's activity log by detecting our own connections. A daemon, in the context of operating systems, is a computer process that runs in the background without direct interaction with users. These processes are started during system boot and continue to run persistently, providing specific services or functionalities.

Chapter 7

Analysis of Collected Data

In this chapter, we will discuss the analysis of the data collected during the execution of Honeytrap. To analyze the data, we need to look at the log files located at /usr/local/etc/honeytrap/honeytrap.log, /opt/honeytrap/attackers.json, /opt/honeytrap/attackers.log, as well as the attacks and downloads stored in the directories /opt/honeytrap/var/honeytrap/attacks and **/opt/honeytrap/var/honeytrap/-downloads**.

The initial data capture period was 6 days, then it was stopped to include modifications and re-run for 20 days. In total, the data capture period was 26 days. Apart from these capture days, it has been executed many times to perform debugging actions.

7.1 Log file analysis. Connection distribution

Observing the activity log or the main log of the honeypot initially does not show any activity with the NFQ execution mode:

```
honeytrap v1.1.0
[2023-06-01 19:08:29] Initializing plugins.
[2023-06-01 19:08:29] ---- Trapping attacks via NFQ. ----
[2023-06-02 10:09:36] ---- honeytrap stopped ----
```

This is due to an incorrect configuration of the NFQ queue and the input rules within the instance itself. Therefore, the following rules have been configured in iptables:

```
1 ubuntu@ip-172-31-43-196:~$ sudo iptables -L
2 Chain INPUT (policy ACCEPT)
3 target     prot opt source               destination
4 ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:smtp
```

```
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:telnet
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:ms-sql-s
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:domain
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:http
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:http-alt
ACCEPT     udp  --  anywhere             anywhere
    udp dpt:tftp
ACCEPT     udp  --  anywhere             anywhere
    udp dpt:mdns
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:8443
ACCEPT     tcp  --  anywhere             anywhere
    tcp dpt:2222

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
```

The application has also been reinstalled, this time configured with the PCAP listening mode. The IPQ listening mode is not recommended as the libipq library is deprecated, and the libnetfilter-queue library is now used instead. Now, incoming activity is detected and logged correctly. However, we quickly notice recurring incorrect activity being detected. A warning is detected during execution, and there is also a frequent error. There is also activity that is correct or at least does not generate errors.

```
[2023-06-02 11:12:59] ---- Trapping attacks on device 'eth0' via PCAP. ----
[2023-06-02 11:13:03] Warning - Process 938 exited on failure.
[2023-06-02 11:13:03] Warning - Process 937 exited on failure.
[2023-06-02 11:13:03] Warning - Process 936 exited on failure.
[2023-06-02 11:13:03] Warning - Process 939 exited on failure.
[2023-06-02 11:13:04] Warning - Process 944 exited on failure.
.....................................................................
[2023-06-02 11:13:14] Error - Protocol 183 is not supported.
[2023-06-02 11:13:14] Error - Protocol 154 is not supported.
[2023-06-02 11:13:15] Error - Protocol 17 is not supported.
[2023-06-02 11:13:15] Error - Protocol 17 is not supported.
.....................................................................
```

The "Protocol not supported" error indicates that the connection uses a protocol that is not supported by the application. According to IANA (Internet Assigned Numbers Authority) [25], the protocol codes that are marked as unsupported are as follows: 17 = UDP, 154 and 183 are marked as Unassigned. It's strange because Honeytrap only supports three protocols: TCP, ICMP, and UDP. This "Error - Protocol 17 is not supported" error is due to the fact that the code in pcapmon.c did not have an option to support UDP in this particular snippet:

```
1     ip = (struct ip_header *) (packet + pcap_offset);
2   if (ip->ip_p == TCP) {
3     tcp   = (struct tcp_header *) ((u_char *) ip + (4 * ip->
     ip_hlen));
4     sport   = ntohs(tcp->th_sport);
5     dport   = ntohs(tcp->th_dport);
6     port_mode = port_flags_tcp[sport] ? port_flags_tcp[sport
     ]->mode : 0;
7   } else if (ip->ip_p == ICMP) {
8     if ((ip = (struct ip_header *) icmp_dissect(ip)) == NULL)
     return;
9     udp   = (struct udp_header *) ((u_char *) ip + (4 * ip->
     ip_hlen));
10    sport   = ntohs(udp->uh_sport);
11    dport   = ntohs(udp->uh_dport);
12    port_mode = port_flags_udp[dport] ? port_flags_udp[dport
     ]->mode : 0;
13  }
14  else {
15    logmsg(LOG_ERR, 1, "Error - Protocol %u is not supported.\
     n", ip->ip_p);
16    return;
17  }
```

After discovering this error in the application code, in a file that I hadn't modified until then, I added the following code snippet:

```
1
2   else if (ip->ip_p == UDP) {
3     udp = (struct udp_header *) ((u_char *) ip + (4 * ip->
     ip_hlen));
4     sport = ntohs(udp->uh_sport);
5     dport = ntohs(udp->uh_dport);
6     port_mode = port_flags_udp[dport] ? port_flags_udp[dport
     ]->mode : 0;
7   }
```

Once this was added, the virtual machine instance was restarted, the application was recompiled, and it was run again. With the collected data, no more UDP errors

were encountered. The other two protocols still appear occasionally, but in this case, it indicates that the application is functioning correctly because it does not allow connections under protocols other than TCP, ICMP, and UDP, and from what we have seen, these are unassigned protocol numbers, therefore not recognized by Honeytrap.

However, a large number of "Warning - Process [pid] exited on failure" are still being detected. This poses a significant problem in data analysis, as 12.3% of the log entries are of this type. These child processes that are executed end up in the EXIT_FAILURE state. To find the cause, I have decided to run the program again with debug-level logging enabled, meaning that debug messages are shown in the log. After doing this, I looked at the log again and found the following:

```
[2023-06-15 23:20:28] 493452 79.52.201.77:55192 requesting tcp connection on 172.31.43.196:22.
[2023-06-15 23:20:28] 493452 Port 22/tcp is configured to be handled in mirror mode.
[2023-06-15 23:20:28] 493452 Calling plugins before dynamic server setup.
[2023-06-15 23:20:28] 493452 172.31.43.196:22 requesting tcp connection on 79.52.201.77:55192.
[2023-06-15 23:20:28] 493452 Port 55192/tcp has no explicit configuration.
[2023-06-15 23:20:28] 493452 Calling plugins before dynamic server setup.
[2023-06-15 23:20:28] 493455 Requesting tcp socket.
[2023-06-15 23:20:28] 493455 Socket created, file descriptor is 13.
[2023-06-15 23:20:28] 493455 Server is now running with user id 1001 and group id 1001.
[2023-06-15 23:20:28] 493455 Listening on port 55192/tcp.
[2023-06-15 23:20:28] 493455 Value of sigpipe[0]: 0
[2023-06-15 23:20:28] 493455 Value of listen_fd: 13
[2023-06-15 23:20:28] 493455 MAX(sigpipe[0], listen_fd): 13
[2023-06-15 23:20:28] 493454 Requesting tcp socket.
[2023-06-15 23:20:28] 493454 Unable to bind to port 22/tcp: Address already in use.
[2023-06-15 23:20:28] 493452 Process 493452 received signal 17 on pipe.
[2023-06-15 23:20:28] 493452 SIGCHILD received.
[2023-06-15 23:20:28] 493452 Process 493454 terminated.
[2023-06-15 23:20:28] 493452 Warning - Process 493454 exited on failure.
```

In the log, it can be observed that a TCP connection request arrives at the virtual machine instance on port 22, which is configured as a mirror port. It's important to note that this is an SSH connection, which is bidirectional. According to the log, a handler is also launched for the remote port when sending packets to the

client. Plugins are called before creating the dynamic server. A socket is created with fd = 13, and the dynamic server is executed with the same user and group ID, which is 1001. It listens on port 55192, which is the port being used by the client. An attempt is made to listen on port 22, but the socket binding fails with the error "Unable to bind to port 22/tcp: Address already in use." The process then receives signal 17 on the pipe, which is the SIGCHLD signal indicating that the child process has terminated. Immediately after that, the warning we are investigating is generated.

Based on this log sequence, it can be inferred that whenever a port is found to be occupied, this warning is generated. In the case you showed, it could be deduced that this warning is generated because you are connected to the instance via SSH, which is the connection being detected. However, this deduction is not entirely correct, as I have tested collecting data and closing the SSH session to ensure that this connection does not interfere. In this new log fragment, the following can be observed:

```
[2023-06-22 09:45:47] 67324  85.209.134.96:55343 requesting tcp connection on 172.31.43.196:23.
[2023-06-22 09:45:47] 67324  Port 23/tcp is configured to be handled in mirror mode.
[2023-06-22 09:45:47] 67324  Calling start_dynamic_server.
[2023-06-22 09:45:47] 67324  Enter on start_dynamic_server.
[2023-06-22 09:45:47] 67324  Calling plugins before dynamic server setup.
[2023-06-22 09:45:47] 67324  Enter on endless loop in start_pcap_mon().
[2023-06-22 09:45:47] 67324  Value of sigpipe[0]: 0
[2023-06-22 09:45:47] 67324  Value of pcap_fd: 12
[2023-06-22 09:45:47] 67324  MAX(sigpipe[0], pcap_fd): 12
[2023-06-22 09:45:47] 78847  Requesting tcp socket.
[2023-06-22 09:45:47] 78847  Socket created, file descriptor is 13.
[2023-06-22 09:45:47] 78847  Server is now running with user id 1001 and group id 1001.
[2023-06-22 09:45:47] 78847  Listening on port 23/tcp.
.............................................................
[2023-06-22 09:47:47] 78847  Case 0:     Timeout.
[2023-06-22 09:47:47] 78847  ->    23/tcp  No incoming connection for 120 seconds - server terminated.
[2023-06-22 09:47:47] 78846  Case 0:     Timeout.
[2023-06-22 09:47:47] 78846  -> 55343/tcp  No incoming connection for 120 seconds - server terminated.
.............................................................
[2023-06-22 09:48:28] 67324  24.237.22.79:21017 requesting tcp connection on 172.31.43.196:23.
[2023-06-22 09:48:28] 67324  Port 23/tcp is configured to be handled in mirror mode.
```

```
22 [2023-06-22 09:48:28] 67324  Calling start_dynamic_server.
23 [2023-06-22 09:48:28] 67324  Enter on start_dynamic_server.
24 [2023-06-22 09:48:28] 67324  Calling plugins before dynamic
     server setup.
25 [2023-06-22 09:48:28] 67324  Enter on endless loop in
     start_pcap_mon().
26 [2023-06-22 09:48:28] 67324  Value of sigpipe[0]: 0
27 [2023-06-22 09:48:28] 67324  Value of pcap_fd: 12
28 [2023-06-22 09:48:28] 67324  MAX(sigpipe[0], pcap_fd): 12
29 [2023-06-22 09:48:28] 85744  Requesting tcp socket.
30 [2023-06-22 09:48:28] 85744  Unable to bind to port 23/tcp:
     Address already in use.
31 [2023-06-22 09:48:28] 67324  Case -1. Error en select().
32 [2023-06-22 09:48:28] 67324  Process 67324 received signal 17
     on pipe.
33 [2023-06-22 09:48:28] 67324  SIGCHILD received.
34 [2023-06-22 09:48:28] 67324  Process 85744 terminated.
35 [2023-06-22 09:48:28] 67324  Warning - Process 85744 exited on
      failure.
36 ............................................................
37 24.237.22.79:21017 requesting tcp connection on
     172.31.43.196:23.
38 [2023-06-22 09:48:36] 67324  Port 23/tcp is configured to be
     handled in mirror mode.
39 [2023-06-22 09:48:36] 67324  Calling start_dynamic_server.
40 [2023-06-22 09:48:36] 67324  Enter on start_dynamic_server.
41 [2023-06-22 09:48:36] 67324  Calling plugins before dynamic
     server setup.
42 [2023-06-22 09:48:36] 67324  Enter on endless loop in
     start_pcap_mon().
43 [2023-06-22 09:48:36] 67324  Value of sigpipe[0]: 0
44 [2023-06-22 09:48:36] 67324  Value of pcap_fd: 12
45 [2023-06-22 09:48:36] 67324  MAX(sigpipe[0], pcap_fd): 12
46 [2023-06-22 09:48:36] 85757  Requesting tcp socket.
47 [2023-06-22 09:48:36] 85757  Unable to bind to port 23/tcp:
     Address already in use.
48 [2023-06-22 09:48:36] 67324  Case -1. Error en select().
49 [2023-06-22 09:48:36] 67324  Process 67324 received signal 17
     on pipe.
50 [2023-06-22 09:48:36] 67324  SIGCHILD received.
51 [2023-06-22 09:48:36] 67324  Process 85757 terminated.
52 [2023-06-22 09:48:36] 67324  Warning - Process 85757 exited on
      failure.
```

Based on the provided analysis, it seems that the error occurs in most cases because there is an active SSH connection and the port cannot be bound because it is already bound to another address. In the second case, a connection is initiated, a socket is created, successfully bound, and listened to. After 120 seconds pass without any packets arriving through the socket, the dynamic server terminates. Here, the error arises that prevents subsequent connections from binding the socket

to the same port. This is because when the dynamic server is closed, the socket is not closed, resulting in a "dead" socket remaining. Therefore, the socket should be closed and changed every time there is an error in connection handling by the dynamic server. The modification would be as follows:

```
1                   switch (select(MAX(connection_fd, sigpipe
    [0]) + 1, &rfds, NULL, NULL, &r_timeout)) {
2               case -1:
3                       if (errno == EINTR) {
4                               if (check_sigpipe() == -1)
    exit(EXIT_FAILURE);
5                               break;
6                       }
7                       logmsg(LOG_ERR, 1, "   %s  Error -
    select() failed: %m.\n", portstr);
8                       close(connection_fd);
9                       return(process_data(attack_string,
    total_bytes, NULL, 0, attack->a_conn.l_port, attack));
10              case 0:
11                      /* no data available, select() timed
    out */
12                      disconnect++;
13                      if (disconnect > 10) {
14                              /* close timeout'd connection
    and process attack string */
15                              logmsg(LOG_INFO, 1, "   %s
    Timeout expired, closing connection.\n", portstr);
16                              close(connection_fd);
17                              return(process_data
18                                      (attack_string,
    total_bytes, NULL, 0, attack->a_conn.l_port, attack));
19                      } else {
20                              if ((send_default_response(
    connection_fd, port, proto, read_timeout)) != -1) {
21                                      logmsg(LOG_ERR, 1,
22                                              "   %s  Error -
     Sending response failed: %m.\n", portstr);
23                                      close(connection_fd);
24                                      return(process_data
25                                              (attack_string
    , total_bytes, NULL, 0, attack->a_conn.l_port, attack));
26                              }
27                      }
28                      break;
29              default:
```

The addition of the ‘close(connection_fd)‘ line in the switch blocks is a step in the right direction to close the socket. However, the application's handling of child processes does not seem appropriate. When a connection is received, it is handled

to create the socket, bind it to the corresponding port, and wait for it to listen.

Additionally, it would be advisable not to open a socket for the remote port in the server's responses. This refers to the situation where an SSH connection is received from a personal computer, for example, from port 57092 to 172.31.43.196:22. When an incoming connection is received, a socket is opened for port 22, which is already in use, resulting in an error. However, when the virtual machine instance sends packets to the personal computer, it opens a socket for the remote port. This behavior does not make sense, so the proposed modification is that if the requesting address is the local or instance address, no socket should be opened.

Regarding the successfully collected connections, in the log of the first execution, we can see that there are 34,245,649 entries in the log. The log records all the activity generated by the honeypot. The initial entries indicate the program initialization and mention that packet trapping is set up on the eth0 interface using PCAP:

```
[2023-06-02 11:12:59] Initializing plugins.
[2023-06-02 11:12:59] Warning - No device given, trying to use
    default device.
[2023-06-02 11:12:59] ---- Trapping attacks on device 'eth0'
   via PCAP. ----
```

‘

These entries provide information about the setup and indicate that the program is ready to capture packets on the eth0 interface using PCAP.

It should be noted that the honeypot executions were recorded over different days, in addition to the times it was executed with debug messages to gather information on the possible cause of the error and warning mentioned earlier. Taking this into account, a total of **645,610** connection attempts were collected over a period of 20 days of monitoring. We will now provide a breakdown of how these connections were distributed. It is worth noting that SSH, SCP, and SFTP connections are not captured correctly, as mentioned before, because if I connect to the virtual machine via SSH, the port 22 will be occupied and any connection attempt will be rejected.

For the HTTP protocol, port 80/tcp is used, and alternatively, port 8080/tcp is used. For the HTTPS protocol, ports 443/tcp and 8443/tcp are used in the same way. In both cases, the first port is the main port, and the second port is an auxiliary port used in web servers.

- On port 80/tcp, 343 connections were established, and none were from localhost. This represents 0.01% of the total connections.

- On port 8080/tcp, 187,983 connections were established, representing 6.02% of the total. It is observed that 185,785 of these connections came from the IP address 127.0.0.1, i.e., from the localhost address. These connections from localhost indicate that a forwarding or proxy mechanism is being used to redirect connections from localhost to port 8080. In each line, it shows that a TCP connection request from 127.0.0.1 (localhost) to 172.31.43.196 on port 8080 is being handled, followed by the line "== 8080/tcp Proxy connection to 127.0.0.1:8080 established," indicating that a proxy connection to localhost on port 8080 has been established. This is unusual and suggests that the proxy mode is misconfigured.
  These connections from localhost are listening to the received packets, which all have a similar size and content, as indicated in the attackers.json file. The content is the default response, and the connection does not carry any payload. The content is as follows:

  ```
  HTTP/1.1 200 OK
  Connection: close
  Date: Sun, 27 Nov 2005 13:07:34 GMT
  Server: Microsoft-IIS/6.0
  X-Powered-By: ASP.NET
  X-AspNet-Version: 2.0.50727
  Accept-Ranges: bytes
  Content-Length: 30
  Cache-Control: private
  Content-Type: text/html; charset=utf-8
  ```

  Connections to port 8080/tcp account for 6.02% of the total, with only real connections (i.e., from a different address than localhost) representing 1.17% or, in other words, 2,198 connections from an IP address other than localhost (127.0.0.1).

- There are 469 connections received on port 443. No connections are received from localhost. These represent 0.02% of the total.

- On port 8443, all captured packets are proxy responses to connections on port 443. Out of the 17,195 detected connections (0.55% of the total), 99.98% of them come from the address 127.0.0.1, indicating that the proxy is being used on this port. Therefore, only 20 connections contain useful information.

Originally, the configuration of these ports was set to proxy mode. Since it has been observed that the proxy option for ports is misconfigured, it has been decided to change all the ports to mirror mode, as indicated in Section 5.4 of Chapter 5.

This distribution of HTTP and HTTPS connections indicates that almost all of them are received from localhost, specifically on ports 8080 and 8443. Out of the 205,990 HTTP or HTTPS connections received, 98.53% of them come from localhost. In other words, around 3,030 connections, representing 1.47%, are from other devices. It is observed that out of the 17,664 HTTPS connections, 97.34% of them are directed to port 8443. The remaining 188,326 connections (91.43%) are HTTP connections, and 99.82% of them are directed to port 8080.

This breakdown of results tells us that HTTP traffic is more frequent than HTTPS, and the interaction with the honeypot is mostly as if it were a web server since ports 8443 and 8080 are commonly used for HTTPS and HTTP servers, respectively.

Regarding incoming connections on port 23 (Telnet), 441,974 connections are established, accounting for 14.17% of the total connections. This represents over two-thirds of the incoming connections. It is significant because it is the only port that was initially set to mirror mode, allowing us to observe this type of traffic without interference from the proxy sending connection requests. The message "23/tcp Error - Sending response failed: Connection Refused" appears in the log a total of 5,159 times. This means that in 1.17% of cases, the response is not sent correctly from the honeypot. Generally, this indicates that either there is no service running on that port or the server is configured to not accept connections on that port. In 436,573 cases, or 98.78%, the server functions correctly, accepts the connection, and sends the response. The content sent is the default response because no content is detected in the incoming packets. In the attackers.log file, the recorded connection that is labeled as an attack does not have an associated payload.

Regarding connections on ports 25/tcp (SMTP) and 1433/tcp (ms-sql-s), they are ignored, and therefore, no packet captures for connections directed to those ports are captured because there are no implemented modules to handle those protocols. It remains to check incoming connections for 69/udp, 53/udp, and 5353/udp.

- 240,945 connections have been established on port 53/udp (DNS), accounting for 7.72% of the total connections. Out of these connections, 225,661 failed to send the response to the client, which is 93.65% of the connections to that port. Out of these failures, 225,370 fail because there is no child process to handle the response, while the remaining failed responses are rejected by the client. The rest of the connections were successful. This port is in normal mode, behaving like a regular DNS server without forwarding a response to each connection. The failure is once again due to a flawed handling of child processes within the application itself.

- 653 connections have been established on port 5353/udp (multicast DNS), representing 0.02% of the total connections. In 100% of the connections, the

Puerto 80/tcp	343	0,01099430508	%
Puerto 8080/tcp	187983	6,025488197	%
Puerto 443/tcp	469	0,01503302939	%
Puerto 8443/tcp	17195	0,5511576555	%
Puerto 25/tcp	0	0	%
Puerto 1433/tcp	0	0	%
Puerto 69/udp	2230232	71,48644607	%
Puerto 20/tcp	0	0	%
Puerto 21/tcp	0	0	%
Puerto 53/udp	240945	7,72309865	%
Puerto 5353/udp	653	0,02093084903	%
Puerto 23/tcp	441974	14,16675508	%
Puerto 22/tcp	3	0,000096160102	%
Total	3119797		

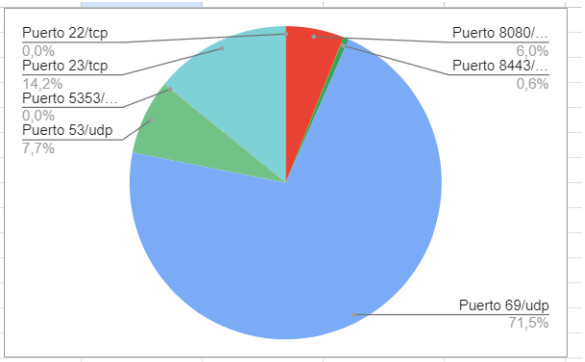


Figure 7.1: Proportion of Connections Graph - First Log

following error is recorded: "5353/udp Error - select() call failed: Bad file descriptor." This error indicates that there are errors in the socket handling by the application.

- 2,230,232 connections have been established on port 69/udp (TFTP), accounting for 71.49% of the total connections. Since it is configured in proxy mode, 1,009 of these connections come from localhost, while the rest are successful but do not receive any payload.

- No connections have been established on port 20/tcp (FTP) and 21/tcp (FTP) because there is no explicit configuration for those ports.

Taking into account all this data, the graph 7.1 is created with the proportion of incoming connections relative to the listening port.

In one of the instances when the application was executed, it was observed in the running processes table that there were zombie processes, as can be seen in this table of open processes marked as "<defunct>".

```
1 user_Se+   35062   17324  0 21:11 ?        00:00:00 /usr/local
    /sbin/honeytrap i eth0 -t 3 -C /usr/local/etc/honeytrap/
    honeytrap.conf
2 ubuntu     35163   35159  0 21:11 ?        00:00:00 scp -f /
    usr/local/etc/honeytrap/honeytrap.log
3 root       35346   17324  0 21:11 ?        00:00:00 [honeytrap
    ] <defunct>
4 root       35347   17324  0 21:11 ?        00:00:00 [honeytrap
    ] <defunct>
5 root       35348   17324  0 21:11 ?        00:00:00 [honeytrap
    ] <defunct>
6 root       35349   17324  0 21:11 ?        00:00:00 [honeytrap
    ] <defunct>
7 root       35350    17324  0 21:11 ?         00:00:00 [
    honeytrap] <defunct>
```

```
root        35351   17324  0 21:11 ?        00:00:00 [honeytrap
    ] <defunct>
root        35352   17324  0 21:11 ?        00:00:00 [honeytrap
    ] <defunct>
user_Se+    37010   17324  0 21:11 ?        00:00:00 /usr/local
    /sbin/honeytrap i eth0 -t 3 -C /usr/local/etc/honeytrap/
    honeytrap.conf
ubuntu      43638   37722  0 21:11 pts/0    00:00:00 grep --
    color=auto honeytrap
```

A zombie process is one that has finished execution but still has an entry in the operating system's process table because its parent process has not yet received its exit status. These "honeytrap" processes have finished execution but have not been fully cleaned up by their parent process. Zombie processes do not consume system resources but can accumulate if not handled properly.

This indicates that the application does not handle the termination of child processes well by the parent process.

7.2 Binary file analysis

Checking the attackers.json file, it can be seen that all registered connections do not have an associated payload, as the length field is marked as 0 and there is no data included in the data_hex field. This is an example entry from the file.

```
{ "is_virtual": false, "@timestamp": "2023-06-23T14:21:21Z", "
    start_time": "2023-06-23T14:21:21Z", "end_time":
    "2023-06-23T14:21:21Z", "attack_connection": { "protocol":
    "tcp", "remote_ip": "172.31.43.196", "remote_port": 60708,
    "local_ip": "169.254.169.254", "local_port": 80, "payload":
     { "md5_hash": "d41d8cd98f00b204e9800998ecf8427e",
"sha512_hash": "
    cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36
ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927

da3e", "length": 0, "data_hex": "" } }, "proxy_connection": {
    "protocol": "ip", "remote_ip": "172.31.43.196", "
    remote_port": 20480, "local_ip": "0.0.0.0", "local_port":
    0, "payload": { "md5_hash": "
    d41d8cd98f00b204e9800998ecf8427e",
"sha512_hash": "
    cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36
ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927

3da3e", "length": 0, "data_hex": "" } }, "operation_mode": 4,
    "download_count": 0, "download_tries": 0, "downloads": { }
    }
```

I have three possible theories as to why no payload content is being recorded. One is that the PCAP capture mode is not functioning correctly. Another is that it may be due to a misconfiguration in the port settings. And the last option is that they could be payload-less connections.
After further examining the honeypot configuration and the source addresses, it is concluded that they are attempts of payload-less connections, such as port scanning. In this case, the attacker may be sending connection request packets without providing additional data. As a result, the length and data fields are empty in the log. In the honeypot instance, there were no vulnerable services that would draw the attacker's attention. By definition, a honeypot only records traffic that interacts with it, so if there is no payload, it can be deduced that the connection is most likely a port scan. Since the attacker doesn't see anything that catches their attention, they don't launch any payload to exploit vulnerabilities. Therefore, vulnerable services will be configured for each listening port. This is detailed in Chapter 5, Section 5.5.

It has been executed again a second time, setting all ports to normal mode because they support well-known protocols to show real behavior to the attacker, and thus test the installed vulnerable services. In a short period of time, 507 incoming SSH connections have been detected. Upon further inspection, the child process fails to send a response to the client initiating the connection with our virtual machine, and for all connections, 12 incoming bytes are detected per incoming packet. This leads us to understand that, in terms of an SSH connection, it could be what is known as a handshake, which is the process by which a client and a server establish a secure SSH connection. During the handshake, security parameters are negotiated, and the client and server mutually authenticate each other. Only incoming packets from the client are detected, as it throws the error mentioned earlier.

Inspecting the directory /opt/honeytrap/var/honeytrap/attacks, it can be seen that its contents are the same as those in the file. The files included in this directory represent each registered attack, but they only contain the default response provided by Honeytrap itself. This occurs for each connection made to localhost when the ports are configured in proxy mode. No files are saved for the rest of the connections because each connection does not have an associated payload, so they cannot be classified as attacks. This can be verified in the file /opt/honeytrap/attackers.json.

In the directory /opt/honeytrap/var/honeytrap/downloads, only one file representing a downloaded file in a connection was found. This file is shown in Figure 7.2.

```
ubuntu@ip-172-31-43-196:~$ ls -l /opt/honeytrap/var/honeytrap/downloads/
total 72
-rw-r--r-- 1 user_Sergio group_tfg 71780 Jun  3 18:06 db6f024d54b8551261213d0d1224c2a6-jklmips
```

Figure 7.2: Suspected Malicious File

This file is an .elf file, which is a standard file format that is common for executable files, object code, shared libraries, and core dumps. Being an executable file and being located in such a directory, it can be considered a malicious file.

To determine its nature and whether it is a malicious file, a static analysis is performed using the following commands.

```
1     ubuntu@ip-172-31-43-196:~$ file /opt/honeytrap/var/
      honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-
      jklmips
2 /opt/honeytrap/var/honeytrap/downloads/
      db6f024d54b8551261213d0d1224c2a6-jklmips: ELF 32-bit MSB
      executable, MIPS, MIPS-I version 1 (SYSV), statically
      linked, stripped
```

The "file" command shows that the file is a 32-bit executable (ELF 32-bit) in the MIPS format (Microprocessor without Interlocked Pipeline Stages). It is statically linked and has been stripped of debugging symbols. This indicates that it is a binary designed to run on devices or embedded systems that use this processor architecture.

```
1     ubuntu@ip-172-31-43-196:~$ strings /opt/honeytrap/var/
      honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-
      jklmips
2   !$
3   !$
4   !$
5 [...]
6 (self
7 /proc/
8 /cmdline
9 /proc
10  !"#$%&'()$%&'()*+,-/dev/watchdog
11 /dev/misc/watchdog
12 /usr/bin
13 ogin
14 enter
15 assword
16 ^\L@\L^
17 R_VJ
18 UJ_MO
19 RUW_I
20 JK[R@[W
21 ST\_YN_^YRSTQ
22 ^YIXSVRPQBI@Y
23 SVSL_UT[
24 YVUO^
25 ]UIR
26 NR[N
```

```
YRST_I_
\[WSVC
UNR_H
N[XV_
IOH_
niUOHY_
T]ST_
kO_HC
JHUY
YW^VST_
_T[XV_
ICIN_W
IR_VV
XOICXUB
xunt
TYUHH_YN
JHUY
M]_N
N\NJ
YOHV
H_XUUN
V@H^
^[HQ
\XR@
WSQ[
YY{~
\\[^
WSJI
WJIV
TSTP[
PO      ^
NIOQOXST
^LHr_VJ_H
lRLkj@
M_OYSX
        XOS
^_V_N_^
xunt
[JJV_N
\UOT^
/proc/net/tcp
/maps
/dev/null
.shstrtab
.init
.text
.fini
.rodata
.ctors
.dtors
.jcr
.data.rel.ro
```

```
.data
.got
.sbss
.bss
.mdebug.abi32
```

The "strings" command returns readable text strings from the binary file. The output of the command has been summarized due to its extensive length.

```
    ubuntu@ip-172-31-43-196:~$ objdump -x /opt/honeytrap/var/honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-jklmips

/opt/honeytrap/var/honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-jklmips:     file format elf32-big
/opt/honeytrap/var/honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-jklmips
architecture: UNKNOWN!, flags 0x00000102:
EXEC_P, D_PAGED
start address 0x00400260

Program Header:
    LOAD off    0x00000000 vaddr 0x00400000 paddr 0x00400000 align 2**16
         filesz 0x00010d10 memsz 0x00010d10 flags r-x
    LOAD off    0x00011000 vaddr 0x00451000 paddr 0x00451000 align 2**16
         filesz 0x000005a0 memsz 0x000008f8 flags rw-
   STACK off    0x00000000 vaddr 0x00000000 paddr 0x00000000 align 2**2
         filesz 0x00000000 memsz 0x00000000 flags rwx

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .init         0000008c  00400094  00400094  00000094  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .text         00010340  00400120  00400120  00000120  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  2 .fini         0000005c  00410460  00410460  00010460  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  3 .rodata       00000850  004104c0  004104c0  000104c0  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .ctors        00000008  00451000  00451000  00011000  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  5 .dtors        00000008  00451008  00451008  00011008  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  6 .jcr          00000004  00451010  00451010  00011010  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  7 .data.rel.ro  0000000c  00451014  00451014  00011014  2**2
```

```
                  CONTENTS, ALLOC, LOAD, DATA
  8 .data         000001d0  00451020  00451020  00011020  2**4
                  CONTENTS, ALLOC, LOAD, DATA
  9 .got          000003b0  004511f0  004511f0  000111f0  2**4
                  CONTENTS, ALLOC, LOAD, DATA
 10 .sbss         0000001c  004515a0  004515a0  000115a0  2**2
                  ALLOC
 11 .bss          00000338  004515c0  004515c0  000115a0  2**4
                  ALLOC
 12 .mdebug.abi32 00000000  0000072c  0000072c  000115a0  2**0
                  CONTENTS, READONLY
SYMBOL TABLE:
no symbols
```

The command 'objdump' provides detailed information about the binary file. Here is the analysis of the output you provided:

- File format: The file has a 32-bit ELF format with big-endian byte order.
- Architecture: The architecture of the file is unknown, as the command could not determine it.
- Start address: The program's start address is 0x00400260.
- Program Header: The file has two load segments (LOAD). The first segment has executable and read-only permissions (flags r-x), while the second segment has read and write permissions (flags rw-).
- Sections: The file has several sections that contain different types of data and code. Some notable sections are: .init, .text, .fini, .rodata, .data, .got, .sbss, and .bss. These sections contain executable instructions, read-only data, modifiable data, and uninitialized data.
- Symbol Table: No symbols have been found in the file's symbol table. This means that no information is provided about the functions or variables defined in the binary.

In conclusion, the file has typical sections of an executable and is intended to be executed on an unknown architecture.

```
    ubuntu@ip-172-31-43-196:~$ readelf -a /opt/honeytrap/var/
   honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-
   jklmips
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, big
   endian
```

```
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           MIPS R3000
  Version:                           0x1
  Entry point address:               0x400260
  Start of program headers:          52 (bytes into file)
  Start of section headers:          71180 (bytes into file)
  Flags:                             0x1007, noreorder, pic,
   cpic, o32, mips1
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         15
  Section header string table index: 14

Section Headers:
  [Nr] Name              Type            Addr     Off    Size
     ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000
   000000 00      0   0  0
  [ 1] .init             PROGBITS        00400094 000094 00008
   c 00  AX  0   0  4
  [ 2] .text             PROGBITS        00400120 000120
   010340 00  AX  0   0 16
  [ 3] .fini             PROGBITS        00410460 010460 00005
   c 00  AX  0   0  4
  [ 4] .rodata           PROGBITS        004104c0 0104c0
   000850 00   A  0   0 16
  [ 5] .ctors            PROGBITS        00451000 011000
   000008 00  WA  0   0  4
  [ 6] .dtors            PROGBITS        00451008 011008
   000008 00  WA  0   0  4
  [ 7] .jcr              PROGBITS        00451010 011010
   000004 00  WA  0   0  4
  [ 8] .data.rel.ro      PROGBITS        00451014 011014 00000
   c 00  WA  0   0  4
  [ 9] .data             PROGBITS        00451020 011020 0001
   d0 00  WA  0   0 16
  [10] .got              PROGBITS        004511f0 0111f0 0003
   b0 04 WAp  0   0 16
  [11] .sbss             NOBITS          004515a0 0115a0 00001
   c 00 WAp  0   0  4
  [12] .bss              NOBITS          004515c0 0115a0
   000338 00  WA  0   0 16
  [13] .mdebug.abi32     PROGBITS        0000072c 0115a0
   000000 00      0   0  1
  [14] .shstrtab         STRTAB          00000000 0115a0
   000069 00      0   0  1
Key to Flags:
```

```
41   W (write), A (alloc), X (execute), M (merge), S (strings), I
       (info),
42   L (link order), O (extra OS processing required), G (group),
       T (TLS),
43   C (compressed), x (unknown), o (OS specific), E (exclude),
44   D (mbind), p (processor specific)
45
46 There are no section groups in this file.
47
48 Program Headers:
49   Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz
        Flg Align
50   LOAD           0x000000 0x00400000 0x00400000 0x10d10 0
      x10d10 R E 0x10000
51   LOAD           0x011000 0x00451000 0x00451000 0x005a0 0
      x008f8 RW  0x10000
52   GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0
      x00000 RWE 0x4
53
54  Section to Segment mapping:
55   Segment Sections...
56    00     .init .text .fini .rodata
57    01     .ctors .dtors .jcr .data.rel.ro .data .got .sbss .
      bss
58    02
59
60 There is no dynamic section in this file.
61
62 There are no relocations in this file.
63
64 The decoding of unwind sections for machine type MIPS R3000 is
       not currently supported.
65
66 No version information found in this file.
67
68 Static GOT:
69  Canonical gp value: 004591e0
70
71  Reserved entries:
72    Address     Access    Value
73   004511f0 -32752(gp) 00000000
74   004511f4 -32748(gp) 80000000
75
76  Local entries:
77    Address     Access    Value
78   004511f8 -32744(gp) 00450000
79   004511fc -32740(gp) 00410000
80   00451200 -32736(gp) 00400000
81   00451204 -32732(gp) 0040cb50
82   [.......................]
83   00451590 -31824(gp) 0040c13c
84   00451594 -31820(gp) 0040ca80
```

```
85   00451598 -31816(gp) 004515a4
86   0045159c -31812(gp) 004511c8
```

The command "readelf" displays information about the internal structure of the ELF file. The output of the command has been summarized, especially in the Local Entries section, as it was very extensive.

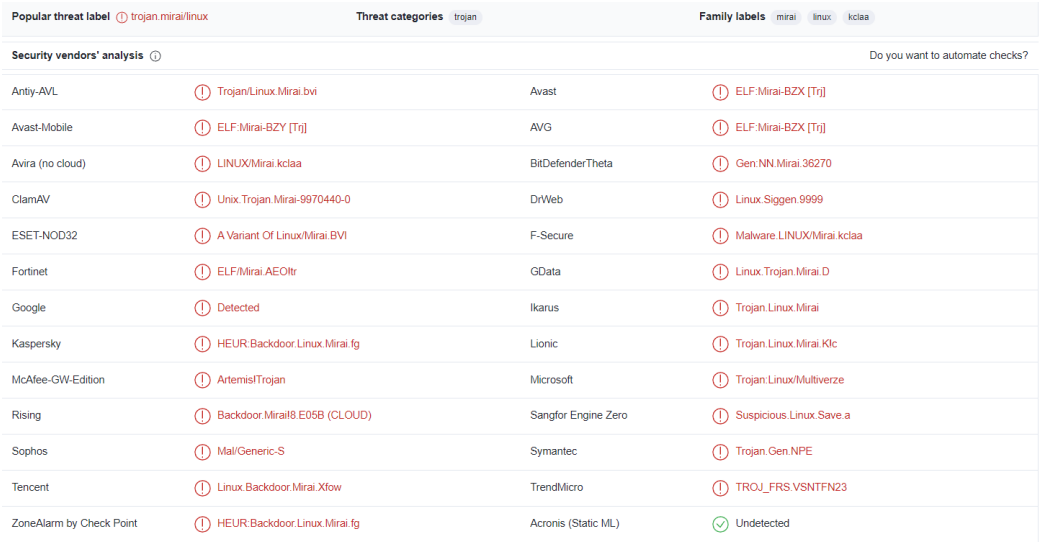
- The "ELF Header" section provides general information about the executable file: its class (ELF32), data type (2's complement, big endian), file version, target machine (MIPS R3000), entry point, etc.
- The "Section Headers" section shows information about the sections present in the executable file: name, type, virtual address, file offset, size, permissions, alignment, etc. Each section has an associated identification number (Nr).
- The "Program Headers" section displays information about the segments present in the executable file. Segments are logical units of an executable file that contain specific information necessary for executing the program. The "LOAD" segment indicates which sections are loaded into memory and their permissions.
- The "Section to Segment mapping" section shows how sections are mapped to segments.
- No dynamic section is found in this file, indicating that dynamic linking is not used. This means that static linking has been performed.
- There are no relocations in this file, which means that no changes have been made to the originally assigned memory addresses.
- Information about the symbol table, local entries, and reserved entries provides details about the symbols defined in the executable file and their associated memory addresses.

```
1    ubuntu@ip-172-31-43-196:~$ sudo strace /opt/honeytrap/var/
     honeytrap/downloads/db6f024d54b8551261213d0d1224c2a6-
     jklmips
2  execve("/opt/honeytrap/var/honeytrap/downloads/
     db6f024d54b8551261213d0d1224c2a6-jklmips", ["/opt/honeytrap
     /var/honeytrap/dow"...], 0x7fffae599a00 /* 13 vars */) = -1
      EACCES (Permission denied)
3  strace: exec: Permission denied
4  +++ exited with 1 +++
5  ubuntu@ip-172-31-43-196:~$ ltrace /opt/honeytrap/var/honeytrap
     /downloads/db6f024d54b8551261213d0d1224c2a6-jklmips
6  "/opt/honeytrap/var/honeytrap/downloads/
     db6f024d54b8551261213d0d1224c2a6-jklmips" is ELF from
     incompatible architecture
```

After performing static analysis, a dynamic analysis is executed on the file. I obtain as a result of this analysis using the strace and ltrace commands that dynamic analysis is not allowed because the file has been statically linked. Therefore, other tools will be used to analyze it and verify if it is indeed a malicious file. By using Radare2, the file's structure is examined, confirming that it is an ELF file, has read and execute permissions, and is an executable file. While there are more characteristics, the most relevant ones are the ones mentioned. Based on these details, it can be deduced that it is a potential malicious file, justifying the use of a honeypot.

7.2.1 Analysis in Virustotal

In order to obtain a more comprehensive analysis, I have scanned this ELF file on the VirusTotal website [26], which is an online platform that provides a free file and URL scanning service to detect malware and other types of threats using a wide variety of antivirus engines and malware detection tools, and I obtained the following results:



Popular threat label trojan.mirai/linux **Threat categories** trojan **Family labels** mirai linux kclaa

Security vendors' analysis Do you want to automate checks?

Antiy-AVL	Trojan/Linux.Mirai.bvi	Avast	ELF:Mirai-BZX [Trj]
Avast-Mobile	ELF:Mirai-BZY [Trj]	AVG	ELF:Mirai-BZX [Trj]
Avira (no cloud)	LINUX/Mirai.kclaa	BitDefenderTheta	Gen:NN.Mirai.36270
ClamAV	Unix.Trojan.Mirai-9970440-0	DrWeb	Linux.Siggen.9999
ESET-NOD32	A Variant Of Linux/Mirai.BVI	F-Secure	Malware.LINUX/Mirai.kclaa
Fortinet	ELF/Mirai.AEOltr	GData	Linux.Trojan.Mirai.D
Google	Detected	Ikarus	Trojan.Linux.Mirai
Kaspersky	HEUR:Backdoor.Linux.Mirai.fg	Lionic	Trojan.Linux.Mirai.Klc
McAfee-GW-Edition	Artemis!Trojan	Microsoft	Trojan:Linux/Multiverze
Rising	Backdoor.Mirai!8.E05B (CLOUD)	Sangfor Engine Zero	Suspicious.Linux.Save.a
Sophos	Mal/Generic-S	Symantec	Trojan.Gen.NPE
Tencent	Linux.Backdoor.Mirai.Xfow	TrendMicro	TROJ_FRS.VSNTFN23
ZoneAlarm by Check Point	HEUR:Backdoor.Linux.Mirai.fg	Acronis (Static ML)	Undetected

Figure 7.3: Virustotal analysis results

It can be seen in the figure 7.3 that according to several search engines, namely 20 of them, this file has been identified as a known threat. Other engines have not been able to detect any threat and several others have not been able to process such a file. Some relevant aspects of the results are detailed below:

- Popular threat label: The label "trojan.mirai/linux" indicates that the scanned file has been identified as a Trojan of the Mirai family designed specifically for Linux operating systems. Trojans are a type of malware that masquerades as legitimate software but performs malicious actions on the affected system.

- Threat categories: The "trojan" category confirms that the file has been classified as a Trojan. Trojans [27] are known to infiltrate a system and allow unauthorized access to attackers, who can use it for various malicious purposes, such as stealing sensitive information or performing additional attacks.

- Family tags: The presence of the tags "mirai" and "linux" indicates that the file is part of the Mirai malware family and specifically targets Linux operating systems. Mirai [28] is a malware family associated with botnet attacks, which primarily target IoT (Internet of Things) devices. These compromised devices are used to carry out massive distributed attacks, which can result in significant disruptions to online infrastructure.

- Security vendor analysis: Analysis results from multiple security vendors show different detections and classifications for the file. Some vendors have specifically identified the file as a variant of Mirai malware or as a Mirai Linux Trojan.

This indicates that the ELF (Executable and Linkable Format) file has been identified as a Mirai Trojan targeting Linux operating systems. The Mirai Trojan, among its various ways of compromising the Linux system, according to the analysis uses the creation of a backdoor to exploit the system. It can install a backdoor on the compromised system, allowing attackers to remotely access and control the system, providing persistent and continuous access to the system even after the original Trojan file has been removed.

7.3 Attackers distribution

I have developed a Python script 8.5.5 that simplifies the /opt/honeytrap/attackers.json file by taking the remote_ip field and storing them in another JSON file. This other file serves as the basis for another script 8.5.5 that will use a free API that allows you to send a request with an IP address and receive geographic information related to that address. In the case of the new remote_ips.json file has many IP addresses, the geolocation script developed on this file is executed. From this execution we find the following results:


```
{
"ip": {
    "220.133.62.88": 6,
    "85.209.134.96": 32,
    "12.230.138.115": 10,
    "195.8.40.172": 10,
    "196.191.162.184": 1,
    "113.118.19.47": 10,
    "194.180.48.128": 17,
    "162.216.150.216": 1,
    "43.154.128.189": 4,
    "122.199.120.87": 10,
```

```
        "42.234.162.44": 10,
        "18.236.135.177": 14,
        "209.97.152.248": 1,
        "178.54.137.92": 10,
        "117.222.235.221": 10,
        "118.163.113.53": 1,
        "59.126.205.141": 10,
        "117.207.119.227": 10,
        "38.166.146.251": 10,
        "1.29.113.238": 10,
        "78.142.18.220": 1,
        "41.77.208.249": 1,
        "106.41.51.128": 1,
        "64.139.246.176": 2,
        "106.41.138.67": 1,
        "91.175.197.35": 1,
        "200.119.227.253": 1,
        "41.86.19.141": 1,
        "117.209.72.28": 1,
        "125.141.72.204": 1,
        "176.97.210.59": 1,
        "121.176.85.158": 1,
        "211.22.185.1": 10,
        "47.37.67.20": 20,
        "41.86.21.11": 20,
        "103.70.83.25": 10,
        "187.71.161.181": 10,
        "12.172.117.103": 1,
        "66.189.122.244": 28,
        "47.116.139.172": 4,
        "120.57.122.21": 2,
        "103.91.180.11": 5,
        "14.167.108.177": 10,
        "91.130.38.170": 6,
        "14.48.241.157": 1,
        "132.247.233.226": 1,
        "190.123.90.180": 1,
        "119.160.197.178": 1,
        "122.179.159.82": 1,
        "133.18.211.209": 1,
        "120.224.118.147": 1,
        "103.110.8.34": 1
    },
    "country": {
        "Taiwan": 27,
        "United States": 126,
        "Ukraine": 20,
        "Ethiopia": 1,
        "China": 37,
        "Hong Kong": 4,
        "South Korea": 13,
        "India": 40,
```

```
        "Venezuela": 10,
        "Bulgaria": 1,
        "Cameroon": 1,
        "France": 1,
        "Chile": 1,
        "Liberia": 21,
        "Germany": 1,
        "Brazil": 10,
        "Vietnam": 10,
        "Sweden": 6,
        "Mexico": 1,
        "Argentina": 1,
        "Japan": 1,
        "Indonesia": 1
    },
    "continent_code": {
        "AS": 133,
        "NA": 127,
        "EU": 29,
        "AF": 23,
        "SA": 22
    },
    "city": {
        "Taichung": 6,
        "Ashburn": 49,
        "Greenwood": 10,
        "Kyiv": 10,
        "Addis Ababa": 1,
        "Shenzhen": 10,
        "North Charleston": 1,
        "Central": 4,
        "Gumi": 10,
        "Anyang": 10,
        "Boardman": 14,
        "Clifton": 1,
        "Bila Tserkva": 10,
        "Ambala": 10,
        "New Taipei": 1,
        "Andong": 10,
        "Dharmavaram": 10,
        "Maracaibo": 10,
        "Hohhot": 10,
        "Dobrich": 1,
        "Yaound\u00e9": 1,
        "Jilin City": 2,
        "Statesboro": 2,
        "Calais": 1,
        "Puente Alto": 1,
        "Buchanan": 1,
        "Tezpur": 1,
        "Siheung-si": 1,
        "Frankfurt am Main": 1,
```

```
        "Namhae-gun": 1,
        "Tainan City": 10,
        "North Richland Hills": 20,
        "Paynesville": 20,
        "Delhi": 12,
        "Porto Alegre": 10,
        "Bardstown": 1,
        "Dudley": 28,
        "Shanghai": 4,
        "Bengaluru": 5,
        "Hanoi": 10,
        "Norsborg": 6,
        "Jeju City": 1,
        "Alvaro Obregon": 1,
        "C\u00f3rdoba": 1,
        "Rajkot": 1,
        "Mumbai": 1,
        "Osaka": 1,
        "": 1,
        "Ancol Timur": 1
    },
    "region": {
        "Taichung City": 6,
        "Virginia": 49,
        "Mississippi": 10,
        "Kyiv City": 10,
        "Addis Ababa": 1,
        "Guangdong": 10,
        "South Carolina": 1,
        "Central and Western District": 4,
        "Gyeongsangbuk-do": 10,
        "Henan": 10,
        "Oregon": 14,
        "New Jersey": 1,
        "Kyiv Oblast": 10,
        "Haryana": 10,
        "New Taipei": 1,
        "Changhua": 10,
        "Andhra Pradesh": 10,
        "Zulia": 10,
        "Inner Mongolia Autonomous Region": 10,
        "Dobrich": 1,
        "Centre": 1,
        "Jilin": 2,
        "Georgia": 2,
        "Hauts-de-France": 1,
        "Santiago Metropolitan": 1,
        "Grand Bassa County": 1,
        "Assam": 1,
        "Gyeonggi-do": 1,
        "Hesse": 1,
        "Gyeongsangnam-do": 1,
```

```
        "Tainan": 10,
        "Texas": 20,
        "Montserrado County": 20,
        "National Capital Territory of Delhi": 12,
        "Rio Grande do Sul": 10,
        "Kentucky": 1,
        "Massachusetts": 28,
        "Shanghai": 4,
        "Karnataka": 5,
        "Hanoi": 10,
        "Stockholm County": 6,
        "Jeju-do": 1,
        "Mexico City": 1,
        "Cordoba": 1,
        "Gujarat": 1,
        "Maharashtra": 1,
        "\u014csaka": 1,
        "Shandong": 1,
        "West Java": 1
    },
    "latitude": {
        "24.144": 6,
        "39.0019": 32,
        "33.5153": 10,
        "50.458": 10,
        "9.026": 1,
        "22.5559": 10,
        "39.0814": 17,
        "32.8608": 1,
        "22.2908": 4,
        "36.1129": 10,
        "36.096": 10,
        "45.8234": 14,
        "40.8364": 1,
        "49.8008": 10,
        "30.3557": 10,
        "24.9466": 1,
        "24.0454": 10,
        "14.4109": 10,
        "10.6666": 10,
        "40.812": 10,
        "43.5606": 1,
        "3.8661": 1,
        "43.8506": 2,
        "32.4413": 2,
        "50.952": 1,
        "-33.6141": 1,
        "5.8797": 1,
        "26.6351": 1,
        "37.3947": 1,
        "50.1103": 1,
        "34.8043": 1,
```

```
        "22.9917": 10,
        "32.8404": 20,
        "6.275": 20,
        "28.6542": 12,
        "-30.0273": 10,
        "37.8065": 1,
        "42.0491": 28,
        "31.2222": 4,
        "12.9634": 5,
        "21.0292": 10,
        "59.2539": 6,
        "33.5109": 1,
        "19.3624": 1,
        "-31.429": 1,
        "22.2904": 1,
        "19.0748": 1,
        "34.6946": 1,
        "36.1155": 1,
        "-6.9344": 1
    },
    "longitude": {
        "120.6844": 6,
        "-77.4556": 32,
        "-90.168": 10,
        "30.5303": 10,
        "38.7439": 1,
        "114.0577": 10,
        "-77.6443": 17,
        "-79.9746": 1,
        "114.1501": 4,
        "128.3433": 10,
        "114.3828": 10,
        "-119.7257": 14,
        "-74.1403": 1,
        "30.0983": 10,
        "76.8019": 10,
        "121.586": 1,
        "120.5072": 10,
        "77.7267": 10,
        "-71.6124": 10,
        "111.6455": 10,
        "27.8299": 1,
        "11.5154": 1,
        "126.5568": 2,
        "-81.7711": 2,
        "1.8526": 1,
        "-70.5893": 1,
        "-10.0547": 1,
        "92.803": 1,
        "126.7814": 1,
        "8.7147": 1,
        "127.9271": 1,
```

```
        "120.2148": 10,
        "-97.2285": 20,
        "-10.7202": 20,
        "77.2373": 12,
        "-51.2353": 10,
        "-85.4592": 1,
        "-71.8944": 28,
        "121.4581": 4,
        "77.5855": 5,
        "105.8526": 10,
        "17.7853": 6,
        "126.5264": 1,
        "-99.2074": 1,
        "-64.1756": 1,
        "70.7915": 1,
        "72.8856": 1,
        "135.5021": 1,
        "120.3024": 1,
        "107.6153": 1
    },
    "accuracy": {
        "200": 62,
        "20": 98,
        "10": 51,
        "50": 33,
        "5": 36,
        "500": 14,
        "1000": 16,
        "100": 24
    },
    "organization": {
        "AS3462 Data Communication Business Group": 27,
        "AS211252 Delis LLC": 49,
        "AS7018 ATT-INTERNET4": 10,
        "AS48650 Anton Tytiuk": 10,
        "AS24757 Ethiopian Telecommunication Corporation": 1,
        "AS4134 Chinanet": 12,
        "AS396982 GOOGLE-CLOUD-PLATFORM": 1,
        "AS132203 Tencent Building, Kejizhongyi Avenue": 4,
        "AS9981 Saero Network Service LTD": 10,
        "AS4837 CHINA UNICOM China169 Backbone": 20,
        "AS16509 AMAZON-02": 14,
        "AS14061 DIGITALOCEAN-ASN": 1,
        "AS48437 PP Merezha": 10,
        "AS9829 National Internet Backbone": 21,
        "AS61461 Airtek Solutions C.A.": 10,
        "AS208046 ColocationX Ltd.": 1,
        "AS64512 Unknown": 1,
        "AS19108 SUDDENLINK-COMMUNICATIONS": 2,
        "AS12322 Free SAS": 1,
        "AS18822 Manquehuenet": 1,
        "AS37203 LIBTELCO": 21,
```

```
325         "AS4766 Korea Telecom": 3,
326         "AS49581 Tube-Hosting": 1,
327         "AS20115 CHARTER-20115": 48,
328         "AS132116 Ani Network Pvt Ltd": 10,
329         "AS22085 Claro SA": 10,
330         "AS27202 CBK-AS": 1,
331         "AS37963 Hangzhou Alibaba Advertising Co.,Ltd.": 4,
332         "AS17813 Mahanagar Telephone Nigam Limited": 2,
333         "AS55947 Bangalore Broadband Network Pvt Ltd": 5,
334         "AS45899 VNPT Corp": 10,
335         "AS1257 Tele2 SWIPnet": 6,
336         "AS278 Universidad Nacional Autonoma de Mexico": 1,
337         "AS52444 Pogliotti & Pogliotti Construcciones S.A.":
    1,
338         "AS45117 Ishans Network": 1,
339         "AS24560 Bharti Airtel Ltd., Telemedia Services": 1,
340         "AS24282 KAGOYA JAPAN Inc.": 1,
341         "AS24444 Shandong Mobile Communication Company Limited
    ": 1,
342         "AS131717 PT Citra Jelajah Informatika": 1
343     }
344 }
```


Looking at the geolocation data stored in this JSON, we can conclude that most of the attacks come from the United States (126) and Asian countries such as India(40) or China(37). This shows that connections and attacks are made from anywhere in the world, although they are usually associated with developed countries. As for the distribution by continent, there are more attacks from Asia (139) and from North America (127). With respect to Europe, Africa and South America, a similar number of attacks are observed, with around twenty. Regarding the cities, the distribution is more diversified, with only Ashburn standing out. The same can be said of the regions. Checking the most repeated coordinates, it was found that

- 24.144, 120.6844 is located in a pond in Taiwan, Taichung, North District.
- 39°00'06.8 "N 77°27'20.2 "W is located in the middle of a pond in Ashburn, Virginia, USA.
- 33.5153, -90.168 is located in a residential neighborhood in Greenwood, Mississippi, USA.
- 6°16'30.0 "N 10°43'12.7 "W is located in an open field near Monrovia, Liberia.
- 32°50'25.4 "N 97°13'42.6 "W is located in a park across from a residential neighborhood in North Richland Hills, Texas, USA.

By selecting just a few examples, we can see that there are coordinates that point to residential areas, while other locations seem to have no apparent relevance. It is important to note that the coordinates do not accurately indicate the geographic

location from which the attacks originated, and should not be considered conclusive evidence. Several factors can influence the inaccuracy of the coordinates, such as the allocation and distribution of IP addresses in different regions, the use of technologies such as virtual private networks (VPNs) that mask the real location of the attackers, among others.

In addition, it is important to keep in mind that the precision of the exact coordinates can vary from 20 to 1000 meters. Generally, coordinates associated with residential areas in developed countries can indicate the actual location with some accuracy, as the organization associated with each IP address provides information about the registered owner or Internet Service Provider (ISP) of that IP. However, there have also been documented cases where the location indicates that the attack originated in a geographic area where there is nothing relevant, which may indicate that the attacker has used proxies, virtual private networks (VPNs) or other methods to mask their actual location and make it appear that the attack originated from a different location.

7.4 Duration of the attacks. Techniques used

According to the fields end_time and start_time in the file, calculating with a script 8.5.5 the difference between both fields it is possible to verify that the attacks have a duration of only a couple of seconds, highlighting a series of attacks that take a few seconds more take between 7 and 8 seconds to complete, in particular are a total of 10 times. This difference is striking. In relation to the above, this short duration and the absence of payloads reinforce the idea that this is an attempt at a reconnaissance or exploration attack. In this type of tactic, the attacker seeks to gather information about the target without necessarily performing a destructive or compromising action, in which the attacker is exploring the availability of a particular service or protocol on the target system. This type of scanning or reconnaissance can be the first step in a series of more sophisticated attacks. Attackers often perform these scans to identify potential entry points or vulnerabilities that can be exploited later. It is therefore seen that the majority of recorded attacks are reconnaissance attacks using a port scan.

However, 10 more interesting attacks have been detected, and looking for them in the file /opt/honeytrap/attackers.json one thing is striking and it is the following:


```
{ "is_virtual": false, "@timestamp": "2023-06-14T15:35:48Z",
"start_time": "2023-06-14T15:35:48Z", "end_time": "2023-06-14T15:35:56Z",
"attack_connection": { "protocol": "tcp", "remote_ip": "12.230.138.115",
"remote_port": 47770, "local_ip": "172.31.43.196", "local_port": 23,
"payload": { "md5_hash": "d41d8cd98f00b204e9800998ecf8427e",
"sha512_hash": "cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d
```

36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327af927da
3e", "length": 0, "data_hex": "" } }, "proxy_connection": { "protocol":
"ip", "remote_ip": "112.5.181.249", "remote_port": 5888, "local_ip":
"0.0.0.0", "local_port": 0, "payload": { "md5_hash": "9278d276a344eff0
18539a51b1b62875",
"sha512_hash": "281250c9463c24b75e7f8b6aeb3dbfb7298bc95c4a5bc39a6ebc7fe
301d586ee5b79a2c555594c809b75d4bdedda86c01bc3f2a49b82f4220dc107512e9484
07", "length": 442, "data_hex": "fffd18fffd20fffb03fffd01fffd1ffffb05ff
fb010d0a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a
a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a
2a2a2a2a2a2a2a2a2a2a2a2a0d0a2a20436f70797269676874202863292032303034 2d323
031352048616e677a686f7520483343205465636 82e20436f2e2c204c74642e20416c6c
207269676874732072657365727665642e20202a0d0a2a20576974686f7574207468652
06f776e6572277320707269 6f72207772697474656e20636f6e73656e742c2020202020
202020202020202020202020202020202020202020202020202020202020202a0d0a2a206e6f2
06465636f6d70696c696e67206f722072657665727365 2d656e67696e656572696e6720
7368616c6c20626520616c6c6f7765642e202020202020202020202020202020202020202
0202a0d0a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a
2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2a2
a2a2a2a2a2a2a2a2a2a2a2a2a0d0a0d0a4c6f67696e206661696c65642e0d000d0a" } },
"operation_mode": 4, "download_count": 0, "download_tries": 0,
"downloads": { } }

It serves as an example of a single entry in the example JSON file, as the other 9 are identical. What is striking is that in attack_connection there is no payload associated with the connection, the hashes md5 and sha256 are those of an empty string and could lead us to think that it is another connection that is an attempt to scan. But in the proxy_connection section there is a payload associated. It may seem an incongruence but it does not have to be. Looking specifically we can see the following tactics and techniques that may have been used in this attack, and therefore in the next 9 attacks of the same type:

- Tactic: Remote service exploitation.
  - Technique: The attacker connects to remote port 23 (Telnet protocol) at local IP address "172.31.43.196" from remote IP address "12.230.138.115" and remote port 47770.
- Tactic: Use of a proxy.
  - Technique: The attacker establishes a connection from remote IP address "112.5.181.249" to remote port 5888, using the IP protocol as a proxy.
- Tactic: Unauthorized access attempt.

  - Technique: The attacker sends a payload to the Telnet service on remote port 23. The payload is empty, i.e. its length is 0 and it does not contain hexadecimal data.

- Tactic: Use of special characters and control sequences.

  - Technique: The payload sent by the attacker contains a control sequence and special characters encoded in hexadecimal ("fffd18fffd20fffb03fffd01fffd1f..."). These characters may be intended to evade security filters and perform unwanted actions on the target system.

- Tactic: Command injection attempt.

  - Technique: The attacker attempts to inject commands into the Telnet service by sending a hexadecimal encoded text string ("2a20436f7079797269676 8742028632920323030..."). This text string contains "CopyRight (c) 2004-2015 Hangzhou H3C Tech. Co., Ltd. all rights reserved." followed by other commands or substrings that are unreadable.

The content of the payload string has been translated from hexadecimal to text string and the following content has been obtained: Attention is drawn to the first

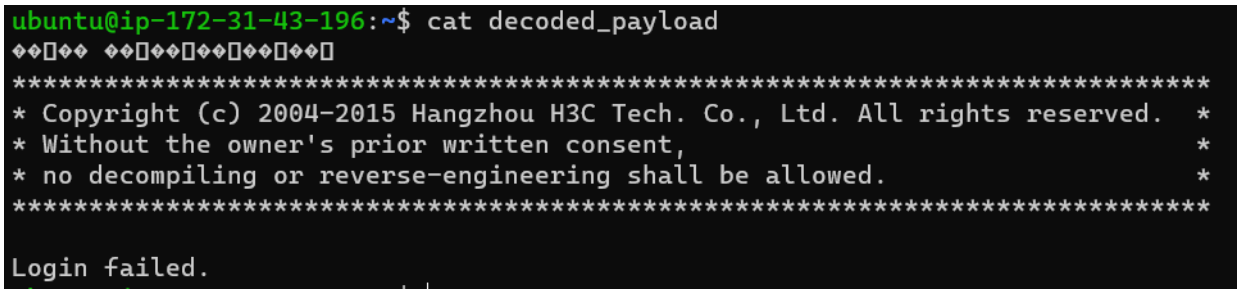


Figure 7.4: Hexadecimal to string

characters that are not readable, this is because the initial value of the string "fffd" represents the Unicode character U+FFFD, which is used to replace invalid or unreadable characters in Unicode. The frequent presence of "fffd" in the hexadecimal code suggests that the data may be corrupted or that some kind of error has occurred during encoding. But there is also another possible option as to why the data is not readable because it has been deliberately obfuscated by encoding the data in a non-standard format, in order to hide possible malicious activity.

Turning to the rest of the content and to summarize we see that an attacker has presumably made a connection from the United States from an IP address (12.230.138.115), which according to Virustotal already has a history of malicious activity, to port 23 to exploit vulnerabilities in the Telnet protocol, which is an old and vulnerable protocol. We know that the attacker has used a proxy to hide

his trail, the IP address of that proxy and the content of the payload associated with the proxy indicate that the attack actually took place in China. The use of a proxy indicates that the attacker has tried to hide his trail. The absence of payload in the attack connection but the presence of the payload in the proxy connection indicates that the attacker has used tunneling [**?**], which is a technique in which malicious traffic or payload is hidden inside another protocol or legitimate traffic to avoid detection, thus making it difficult to detect such traffic. This may explain the presence of malware in the system as we have seen above that was not detected during the execution of the honeypot.

Once we have seen these striking attacks, we can summarize that most attacks do not use a proxy nor do they have an associated payload. This may indicate that either there are errors in the packet capture, which seems to be the case according to my checks, or that some of the following techniques are being used:

- Port scanning attacks: Port scanning attacks are attempts to identify open ports on a system or network. Attackers perform scans by sending requests to different ports to determine which services or applications are available. These attacks usually do not have an associated payload, as their main objective is to identify potential entry points or vulnerabilities.

- Brute-force or dictionary attacks: In these attacks, attackers attempt to guess passwords or keys by trying different combinations of words or characters. No specific payload is needed, as attackers send multiple authentication attempts using common or automatically generated password combinations. This may be occurring in attacks on port 22 and port 23.

- Denial of Service (DoS) or flooding attacks: In these attacks, the goal is to saturate a system or network with a large amount of malicious requests or traffic to cause resources to be exhausted and the system to become inaccessible. These attacks do not require an associated payload, as they rely on the volume or intensity of malicious requests to achieve their goal.

- Probes or vulnerability scans: Attackers can perform automated scans for systems or applications with known vulnerabilities. These attacks involve searching for known weaknesses in systems or applications without the need for a specific payload.

Regarding the problems mentioned in the data capture are still present after many tests, it is due to an incorrect management of the events that arrive as new connections. In the case of TCP, when the connection is established, packets are sent with the data to be sent through the connection. From what I have seen while debugging these new packets are often treated as a new connection. This is a serious error because it prevents to read from the socket that was initially associated to the address that initiated the connection.

Chapter 8

Conclusions

In this last chapter we will discuss the conclusions drawn from the work as well as the ideas to improve this project in the future. But before beginning to expose the conclusions it is necessary to comment that the chapters 1, 4, 5 and 6 are totally original, as well as the code that has been commented in the section. The chapters 2 and 3 have been taken bibliographical references, as well as in the first annex. Regarding the code, it has been previously discussed in the 3.9 section of the chapter 3. And in the last chapter the data analysis is original, taking bibliographic references for the explanations of some terms.

After reviewing the log files and mentioned directories, several conclusions can be drawn. The first and, in my opinion, most important conclusion is that I have been working with an outdated and obsolete tool, with several code errors that hinder its proper functioning. On paper, Honeytrap seemed like a suitable tool for the reasons stated, but when executing the honeypot, it became evident that it is an outdated application with a handling of child processes that affects performance, proper execution, packet content capture, and clear analysis of the results. This is my fault for incorrectly calibrating the choice of the correct honeypot.

The second conclusion I draw is regarding the distribution of connections on the listened ports. Attackers tend to establish connections to ports that support less secure protocols and have fewer security mechanisms to protect the connections, making it easier to exploit systems. This explains why protocols like Telnet, TFTP, or HTTP dominate the majority of connections compared to similarly functional but more secure protocols such as SSH, FTP, or HTTPS.

Regarding the presence of many connection attempts that are accepted but do not transmit payload, I interpret this as the attacker performing port scanning with a tool like nmap to check the status of the ports. The virtual machine then responds, but if the port in the honeypot is not configured as normal and reflects the entire connection as in mirror mode or forwards the connections to a proxy server in

proxy mode, it may give the attacker the impression that a honeypot exists on the virtual machine when encountering behavior that may seem unusual in well-known protocols, and they may decide not to interact with the honeypot. Therefore, the optimal configuration mode for the ports is normal mode to make the existence of the honeypot unnoticed.

A honeypot deployed in an environment that has no apparent vulnerabilities and operates unusually in certain protocols is an unappealing target for exploitation. For a honeypot to be attractive and record malicious activity, it requires an optimal configuration in the behavior of the ports and having known but not evident vulnerabilities.

By definition, every connection to a honeypot is suspicious since only direct interactions with it are recorded. Therefore, if there is evidence of suspicious connections, even though there is little evidence that these are malicious interactions.

Once we have seen the probable techniques that have been used, we can conclude that the most commonly used technique is port scanning, there is evidence that tunneling has been used to exploit vulnerabilities and there are also suspicions of brute force attacks using dictionaries. This indicates that attacks on the environment where honeypot is deployed try to be as stealthy as possible to leave no trace by hiding the origin of the attack with the use of a proxy and to hide the potential presence of malware in the connections by obfuscating the data. Other attackers only investigate the system for vulnerabilities and try to brute force access to it.

The expected results regarding the presence of malicious activities have not been obtained. However, a distribution of observed traffic similar to the original prediction has been obtained.

8.1 Future Work

In this section, some ideas and proposals to improve Honeytrap, as it is open source, will be detailed.

The first future line of work to consider is maintaining the code to keep it up to date and offer functionality that is currently obsolete and causing errors, such as the NFQ and IPQ listening modes.

A modification of the htm_dnsDetection module is also proposed to add relative functionality so that the Honeytrap application can detect DNS responses in an intercepted network packet, providing a more comprehensive and clear understanding of the DNS communication that can be intercepted in network traffic.

Other proposals to improve and add functionality to this application include adding modules that provide functionality for the STMP and MySQL protocols, as the ports associated with these protocols are considered in the "Ignore" section of the configuration file. It would also be interesting to improve the code for SSH, SFTP, and SCP.

One last modification for the future and perhaps the most important for the correct functioning of the application of all is to modify the management of incoming events when a TCP connection is received and packets are received again within that connection.

Appendix I

Tools and Technologies Used in the Project

Overleaf

Overleaf was used to write the project's report. It is a collaborative online LaTeX editor based on the cloud, used for creating, editing, and publishing scientific and academic documents in real time.

A template with the main structure of a thesis was used to write the report. For better organization of the content, it was divided into several .tex files, each corresponding to a chapter of the thesis. Various LaTeX packages were also used to improve the writing and presentation of the report.

Figure 8.1 shows the organization of the files to create the LaTeX document.

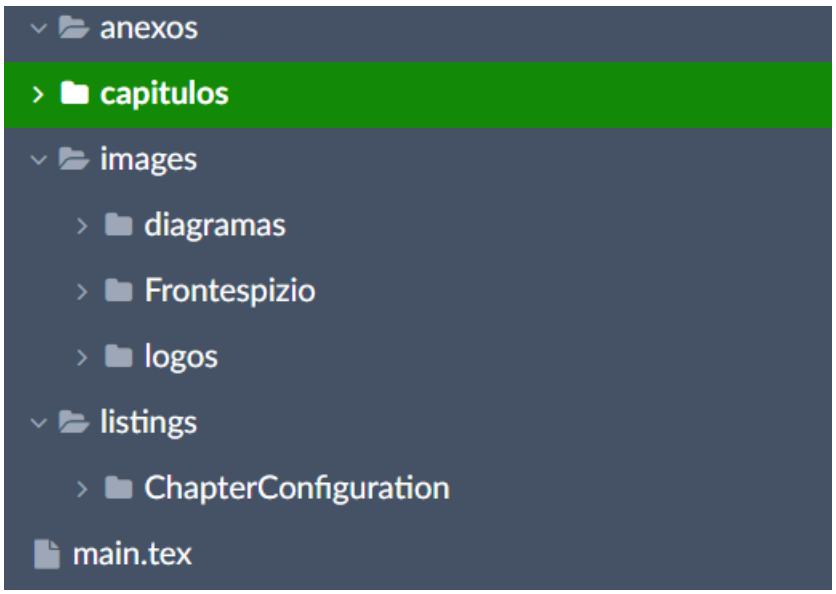


Figure 8.1: LaTeX File Organization

AWS

AWS or Amazon Web Services [29] is the chosen hosting platform for deploying the honeypot in the network. AWS is a cloud services platform developed by Amazon that provides a wide range of hosting, management, and scalability services for online applications and services.

AWS offers the following features:

- Scalability and flexibility: It allows scaling hosting resources according to the project's needs. Storage, processing power, and other resources can be dynamically increased or decreased.
- Wide range of services such as virtual servers (EC2), cloud storage (S3), databases (RDS), networking services (VPC), load balancers, among others.
- Reliability and availability: AWS has a robust and highly available architecture across multiple geographical regions. This ensures that applications and resources are available even in the event of hardware failures or other issues in specific regions.
- Security: It provides advanced security measures to protect data and applications hosted on the platform. It offers encryption options, access management tools, firewalls, and other security measures to ensure data availability and integrity.
- Ease of use: It offers a web-based management interface, a command-line interface, and various SDKs to facilitate resource management.

Amazon Elastic Compute Cloud (EC2)

Amazon Elastic Compute Cloud [30] is the AWS service that provides scalable capacity and flexibility in the cloud. It allows users to obtain and configure virtual computing capacity in the cloud.

EC2 offers the following key features:

- Virtual machine instances: It allows users to create, configure, and launch virtual machine instances based on a variety of operating systems such as macOS, Windows, or Linux. Instances are scalable and can be configured with different resource sizes.
- Instance type selection: A wide range of instance types is available, each optimized for specific use cases and different workloads.
- Automatic scalability: It enables automatic scaling of the number of instances using Autoscaling, which can decrease or increase the number of instances based on predefined policies and various metrics.

- Flexible storage: EC2 offers various storage options, allowing users to adapt to different storage requirements depending on the application or use case.
- Security and management: It provides robust security policies such as the use of key pairs for instance access, the use of security groups to control network traffic, and integration with AWS Identity and Access Management (IAM) to manage access control permissions.
- Integration with other AWS services: EC2 integrates with many other services provided by AWS, allowing users to leverage a wide range of tools and capabilities.

In summary, by using EC2, users can easily create, configure, and manage virtual machine instances to run applications and services, adapting to their performance, scalability, and storage requirements as needed.

Elastic Load Balancer (ELB)

Elastic Load Balancer (ELB) [31] is a load balancing service provided by AWS that automatically distributes traffic across multiple AWS instances. Its main objective is to improve the availability and scalability of web services. Some of its key features include:

- Load distribution: ELB distributes incoming traffic among multiple EC2 instances within the same region. It balances the workload across instances, preventing a single instance from being overwhelmed, thereby improving performance and availability.
- Intelligent load balancing: It uses load balancing algorithms to distribute traffic more intelligently. It can utilize load-based balancing, taking into account the current load on instances, or a more traditional round-robin approach.
- SSL/TLS: ELB allows the configuration of SSL/TLS encryption to protect the traffic between clients and EC2 instances. It ensures secure communication and the privacy and integrity of transmitted data.
- Integration with other AWS services: ELB integrates with many other services provided by AWS, enabling users to leverage a wide range of tools and capabilities.

By utilizing ELB, users can achieve improved performance, scalability, and availability for their web services by effectively distributing traffic across multiple instances.

Visual Studio Code

The code editor used to add and implement the required functionalities was Visual Studio Code [32]. It is a code editor developed by Microsoft and offers the following features [33]:

- Lightweight and fast: It supports a wide range of programming languages and allows for easy code creation, editing, and debugging with various tools readily available. This makes it perfect for daily use.
- Customizable: There is a wide variety of easily installable extensions and integrations that can be used within the code editor.
- Robust architecture: Visual Studio Code combines the best of web technologies, native apps, and specific programming languages. By using Electron, it combines web technologies like JavaScript and Node.js with the speed and flexibility of native apps.
- Cross-platform: It is available on different platforms such as macOS, Linux, and Windows.

Astah

Astah is the software that has been used for the design phase, specifically when working with the domain model. It allows for quick visualization of diagrams to better understand the analysis and design stages of software development.

Astah Professional enables the modeling of UML diagrams such as entity-relationship (E/R) diagrams, class diagrams, flowcharts, mind maps, and more. The interface is user-friendly and has a relatively easy learning curve. It features lightweight technology that allows for the quick creation of these diagrams.

8.2 Virustotal

Virustotal [26] is a free online service that provides a scan of files and URLs for potential malware threats. It was developed by Hispasec Sistemas, a computer security company based in Spain.

Virustotal's platform allows users to upload files or enter URLs for scanning using a wide range of antivirus engines and malware detection tools. Currently, Virustotal has more than 70 antivirus engines in its suite, which includes commercial solutions and open source antivirus engines.

Once a file or URL is submitted to Virustotal, a thorough scan is performed using the available antivirus engines. The result of the scan is presented in the form of a report, which shows the results of each antivirus engine used and any threat detections found. This can help users determine whether a specific file or URL is malicious or potentially dangerous.

Appendix II

8.3 htm_httpDownload.c

8.3.1 Function is_https()

```
int is_https(char *request){
  if(strcmp(request, "https://")){
    return 1; // is HTTPS
  }
  else if(strcmp(request, "http://")){
    return 0; // is HTTPS
  }
  else{
    return -1; // not valid request
  }
}
```

8.3.2 Modification on cmd_parse_for_http_url()

```
//for https
    else if((attack->a_conn.payload.size -i >= 8) && (memcmp(
   string_for_processing + i, "https://", 8) == 0)){
      is_https = 1;
      start = string_for_processing+i;

      /* 0-terminate URL */
      for (end = start, j=0; j<strlen(start) && end[0]; end =
   &start[j++]) {
        if (isspace(end[0])) end[0] = 0;
        else if (!isprint(end[0])) end[0] = 0;
      }
      if (isspace(end[0])) end[0] = 0;

      logmsg(LOG_DEBUG, 1, "HTTPS download - URL found: '%s'\n
   ", start);

      // increase number of download tries
```

```
16        attack->dl_tries++;
17
18        /* assemble wget download command and execute it */
19        if (asprintf(&cmd, "%s %s %s -q", https_program,
    https_options, start) == -1) {
20          logmsg(LOG_ERR, 1, "HTTP download error - Unable to
    allocate memory: %s.\n", strerror(errno));
21          free(string_for_processing);
22          free(cmd);
23          return(-1);
24        }
25        logmsg(LOG_DEBUG, 1, "HTTPS download - Calling '%s'.\n",
     cmd);
26        if ((f = popen(cmd, "r")) == NULL) {
27          logmsg(LOG_ERR, 1, "HTTPS download error - Cannot call
     download command: %m.\n");
28          free(string_for_processing);
29          free(cmd);
30          return(0);
31        }
32
33        total_bytes = 0;
34        do {
35          if ((binary_stream = realloc(binary_stream,
    total_bytes + BUFSIZ)) == NULL) {
36            logmsg(LOG_ERR, 1, "HTTPS download error - Unable to
     allocate memory: %s.\n", strerror(errno));
37            pclose(f);
38            free(string_for_processing);
39            free(cmd);
40            free(binary_stream);
41            return(-1);
42          }
43          new_bytes = fread(binary_stream + total_bytes, 1,
    BUFSIZ, f);
44          total_bytes += new_bytes;
45        } while (!feof(f));
46
47        if (ferror(f)) {
48          logmsg(LOG_ERR, 1, "HTTP download error - Unable to
    allocate memory: %s.\n", strerror(errno));
49          pclose(f);
50          free(string_for_processing);
51          free(cmd);
52          free(binary_stream);
53          return(-1);
54        }
55
56        // add download to attack record
57        if (total_bytes) {
58          logmsg(LOG_DEBUG, 1, "HTTPS download - Adding download
     to attack record.\n");
```

```
59
60          add_download("https", TCP, 0, 0, NULL, NULL, strrchr(
    start, '/')+1, start, binary_stream, total_bytes, attack);
61
62          logmsg(LOG_INFO, 1, "HTTPS download - %s successfully
    downloaded and attached to attack record.\n", start);
63       } else logmsg(LOG_INFO, 1, "HTTPS download - No data
    received.\n");
64
65       pclose(f);
66       free(cmd);
67       free(binary_stream);
68
69       i += strlen(start);
70     }
```

8.4 htm_dnsDetection.c

8.4.1 Function is_dns_query()

```
1  /**
2
3  This function checks whether a given packet is a valid DNS
       query or response.
4
5  It first checks if the packet has the minimum size required to
        be a valid DNS packet.
6
7  Then it checks the value of the query/response bit to
       determine if it's a query or a response.
8
9  @param packet A pointer to the packet data.
10
11 @return -1 if the packet size is invalid, 1 if it's a query, 0
        if it's a response.
12 */
13 int is_dns_query(char packet){
14   //Check if the packet has the minimum size to be a valid DNS
        packet
15   if(packet[2] < 0x01 || packet[2] > 0xff || packet[3] < 0x01
       || packet[3] > 0xff){
16     return -1;
17   }
18
19   // Extract the length of the question section from the
       packet
20     int questionLength = (packet[2] << 8) | packet[3];
21
22     // Check if the packet size is at least equal to the
```

```
    length of the question section
23     if (questionLength > 0 && questionLength <= (packetSize -
    HEADER_SIZE)) {
24         // Check the value of the query/response bit
25         if ((packet[2] & 0x80) == 0x00) {
26             return 1; // It's a query
27         } else {
28             return 0; // It's a response
29         }
30     }
31 }
```

8.4.2 Function cmd_parse_for_dns_query()

```
1  int cmd_parse_for_dns_query(Attack *attack, struct dns_query *
    query){
2     /*
3     Format of DNS query:
4     <dd-mmm-YYYY HH:MM:SS.uuu> <client IP>#<port> query: <
    query_Domain name> <class name> <type name> <- or +>[SETDC]
5     <(name server ip)>
6     */
7
8     char *date = NULL, *hour = NULL, *client_ip = NULL, *port
    = NULL, *query_str = NULL, *domain_name = NULL,
9          *class_name = NULL, *type_name = NULL, *status = NULL
    , *name_server_ip = NULL;
10
11   // Provisionales
12     FILE *f = NULL;
13     char *start = NULL, *end = NULL, *cmd = NULL;
14     u_char *binary_stream = NULL;
15
16     char *payload = attack->a_conn.payload.data;
17     size_t payload_size = attack->a_conn.payload.size;
18
19    /* no data - nothing todo */
20   if ((attack->a_conn.payload.size == 0) || (attack->a_conn.
    payload.data == NULL)) {
21     logmsg(LOG_DEBUG, 1, "DNS connection - No data received.\n
    ");
22     return(0);
23   }
24
25     logmsg(LOG_DEBUG,1, "DNS connection - Parsing attacking
    string (%d bytes) for DNS querys", attack->a_conn.payload.
    size);
26
27     string_for_processing = (char *) malloc(attack->a_conn.
```

```
 payload.size + 1);
memcpy(string_for_processing, attack->a_conn.payload.data,
 attack->a_conn.payload.size);
string_for_processing[attack->a_conn.payload.size] = 0;

  /*for(int i = 0; i < attack.a_conn.payload.size; i++){
      parse_string = attack_string +1;
      *token = strtok(string_for_processing, " ");
      if(token[i])
  }*/

// extract user and host tokens
  token = strtok(string_for_proccesing, " ");

  if (token != NULL) {
    date = token;
    token = strtok(NULL, " ");
    if (token != NULL) {
        hour = token;
        // extract command token
        token = strtok(NULL, "");
        if (token != NULL) {
          client_ip = token;
      port = strtok(client_ip, "#");
        }
    }
  }

token = strtok(NULL, " ");
if( token != NULL){
  query_str = token;
}

token = strtok(NULL, " ");
if( token != NULL){
  domain_name = token;
}

token = strtok(NULL, " ");
if( token != NULL){
  class_name = token;
}

token = strtok(NULL, " ");
if( token != NULL){
  type_name = token;
}

token = strtok(NULL, " ");
if( token != NULL){
  status = token;
}
```

```
78
79    token = strtok(NULL, " ");
80    if( token != NULL){
81      name_server_ip = token;
82    }
83
84    free(string_for_processing);
85
86      /* add dns connection to attack record */
87    if (total_bytes) {
88      logmsg(LOG_DEBUG, 1, "DNS connection - Adding connection
       to attack record.\n");
89      add_query(date, hour, client_ip, port, domain_name,
       class_name, type_name, status, name_server_ip, attack);
90
91      logmsg(LOG_NOTICE, 1, "DNS connection - %s attached to
       attack record.\n", save_file);
92    } else {
93      logmsg(LOG_NOISY, 1, "DNS connection - No data received.\n
       ");
94    }
95
96      return 0;
97  }
```

8.5 htm_sshDownload.c

8.5.1 Function cmd_parse_for_ssh()

```
1  int cmd_parse_for_ssh(Attack *attack){
2    int i=0;
3    char *string_for_proccessing;
4    char ssh_str[] = "ssh";
5    struct in_addr *addr = NULL;
6
7    uint32_t size_payload = attack->a_conn.payload.size;
8
9    /* no data - nothing todo */
10   if((size_payload == 0) || (attack->a_conn.payload.data ==
      NULL)){
11     logmsg(LOG_DEBUG, 1, "SSH download - No data received,
      nothing to download.\n");
12     return(0);
13   }
14   logmsg(LOG_DEBUG, 1, "SSH download - Parsing attack string
      (%d bytes) for ssh commands.\n", size_payload);
15
16   string_for_proccessing = (char*) malloc(size_payload + 1);
17   memcpy(string_for_proccessing, attack->a_conn.payload.data,
```

```
    size_payload + 1);
18  string_for_proccessing[size_payload] = 0;
19
20  for(i = 0; i < size_payload; i++){
21    if((size_payload - i >= sizeof(ssh_str)) && (memcmp(
    string_for_proccessing + i, ssh_str, sizeof(ssh_str)) == 0)
    ){
22      logmsg(LOG_DEBUG, 1, "SSH download - SSH command found.\
    n");
23
24      /* do ssh download */
25      addr = (struct in_addr *) &(attack->a_conn.l_addr);
26      get_sshcmd(string_for_proccessing, size_payload, attack)
    ;
27      return(1); //Command ssh found
28    }
29  }
30  logmsg(LOG_DEBUG, 1, "SSH download - No ssh command found.\n
    ");
31
32  free(string_for_proccessing);
33
34  return(0); //Command ssh not found
35 }
```

8.5.2 Function get_sshcmd

```
1  int get_sshcmd(char *attack_string, uint32_t string_size,
     Attack *attack){
2    /*An ssh command looks like this:
3      ssh [options] [user@]host [command]
4    */
5
6    char *token, *rest = attack_string, *user = NULL, *host =
     NULL, *command = NULL, *r_addr = NULL, *r_port = NULL;
7    // format of remote_path = user@host:/path/to/file
8    // format of local_path = /path/to/file
9    char *remote_path = NULL, *local_path = NULL, *conn_type =
     NULL;
10
11   // skip the 'ssh' command token
12   strtok(rest, " ");
13
14
15
16   // extract user and host tokens
17   token = strtok(rest, "@");
18   if (token != NULL) {
19       user = token;
```

```
20        token = strtok(NULL, " ");
21      if (token != NULL) {
22        host = token;
23
24        //get host IP address
25        struct hostent *he;
26        struct in_addr **address_list;
27
28        if((he = gethostbyname(host)) == NULL){
29          logmsg(LOG_ERR, 1, "Error: could not resolve host %s\n
     ", host);
30              return -1;
31        }
32        else{
33          address_list = (struct in_addr **) he->h_addr_list;
34          if(address_list[0] != NULL){
35            r_addr = address_list[0]->s_addr;
36          }
37        }
38      }
39      }
40
41    // extract command token
42    token = strtok(NULL, "");
43    if (token != NULL) {
44      command = token;
45      // extract remote and local paths, if present
46          char *remote_delim = strstr(command, ":");
47          char *local_delim = strrchr(command, '/');
48
49          if (remote_delim != NULL) {
50              remote_path = remote_delim + 1;
51          }
52          if (local_delim != NULL) {
53              local_path = local_delim + 1;
54          }
55    }
56
57    char *filename = NULL;
58
59    if(command != NULL){
60      // check if command is scp or sftp
61        if (strstr(command, "scp") != NULL ){
62        conn_type = "scp";
63        char *c = strstr(command, " ");
64        if(c!=NULL){
65          c++;
66          filename = strrchr(c, '/');
67          if(filename != NULL){
68            filename++;
69          }
70          else{
```

```
71            filename = p;
72          }
73        }
74      }
75      else if(strstr(command, "sftp") != NULL) {
76        conn_type = "sftp";
77        char *c = strstr(command, " ");
78        if(c!=NULL){
79          c++;
80          filename = strrchr(c, '/');
81          if(filename != NULL){
82            filename++;
83          }
84          else{
85            filename = p;
86          }
87        }
88      }
89    }
90
91    /* add ssh connection to attack record */
92      logmsg(LOG_DEBUG, 1, "SSS connection - Adding connection
     to attack record.\n");
93      add_download("SSH", 6, r_addr, r_port, user, NULL,
     filename, remote_path, NULL, NULL a);
94      logmsg(LOG_NOTICE, 1, "SSH connection - %s attached to
     attack record.\n", save_file);
95
96      return get(get_ssh_resource(user, host, remote_path,
     local_path, a, conn_type, filename));
97  }
```

8.5.3 Function get_ssh_resource

```
1  int get_ssh_resource(const char* user, const char* host, const
       char* remote_path, const char* local_path,
2    Attack* attack, const char* conn_type, const char* filename)
      {
3    ssh_session ssh = ssh_new();
4
5    if(ssh == NULL){
6      logmsg(LOG_ERR, 1, "SSH download error - Session cannot be
       established. \n");
7      return -1;
8    }
9
10   ssh_options_set(ssh, SSH_OPTIONS_HOST, host);  // establece
      el host al que se va a conectar
11     ssh_options_set(ssh, SSH_OPTIONS_USER, user);  //
```

```
    establece el usuario con el que se va a conectar
12
13     int status = ssh_connect(ssh);  // realiza la conexi n
    SSH
14     if (status != SSH_OK) {
15         logmsg(LOG_ERR, 1, "SSH download error - Connection
    error with server SSH.\n");
16         ssh_free(ssh);
17         return -1;
18     }
19
20   status = ssh_userauth_publickey_auto(ssh, NULL, NULL);  //
    autentica la conexi n SSH utilizando las claves p blicas
    del usuario
21     if (status != SSH_AUTH_SUCCESS) {
22         logmsg(LOG_ERR, 1, "SSH download error -
    Authentication error.\n");
23         ssh_disconnect(ssh);
24         ssh_free(ssh);
25         return -1;
26     }
27
28   if (conn_type != NULL && strcmp(conn_type, "SFTP") == 0) {
29     get_ssh_resources_by_sftp(user, host, remote_path,
    local_path, a, ssh, filename);
30   } else if (conn_type != NULL && strcmp(conn_type, "SCP") ==
    0) {
31     get_ssh_resources_by_scp(user, host, remote_path,
    local_path, a, ssh, filename);
32   } else {
33     logmsg(LOG_ERR, 1, "Error: unknown connection type %s\n",
    conn_type);
34     return -1;
35   }
36
37   return 1;
38 }
```

8.5.4 Function get_ssh_resource_by_sftp

```
1 int get_ssh_resource_by_sftp(const char* user, const char*
    host, const char* remote_path, const char* local_path,
    Attack* attack, ssh_session ssh){
2   sftp_session sftp = sftp_new(ssh);
3   if (sftp == NULL) {
4         logmsg(LOG_ERR, 1, "SSH download error - SFTP session
    cannot be created.\n");
5         ssh_disconnect(ssh);
6         ssh_free(ssh);
```

```
7           return -1;
8       }
9
10      status = sftp_init(sftp);  // inicializa la sesi n SFTP
11      if (status != SSH_OK) {
12          logmsg(LOG_ERR, 1, "SSH download error - Error
    initializing SFTP session.\n");
13          sftp_free(sftp);
14          ssh_disconnect(ssh);
15          ssh_free(ssh);
16          return -1;
17      }
18
19      sftp_file file = sftp_open(sftp, remote_path, O_RDONLY, 0)
    ;  // abre el archivo remoto en modo lectura
20      if (file == NULL) {
21          logmsg(LOG_ERR, 1, "SSH download error - Error opening
    remote file.\n");
22          sftp_free(sftp);
23          ssh_disconnect(ssh);
24          ssh_free(ssh);
25          return -1;
26      }
27
28  FILE* f = fopen(local_path, "wb");  // abre el archivo local
    en modo escritura binaria
29      if (f == NULL) {
30          logmsg(LOG_ERR, 1, "SSH download error - Error openng
    local file.\n");
31          sftp_close(file);
32          sftp_free(sftp);
33          ssh_disconnect(ssh);
34          ssh_free(ssh);
35          return -1;
36      }
37
38      char buffer[1024];
39  char *data = NULL;
40      int  nbytes= 0, data-size = 0;
41  do {
42          nbytes = sftp_read(file, buffer, sizeof(buffer));
43          if (nbytes > 0) {
44        data_size += nbytes;
45        data = realloc(data, data_size);
46        memcpy(data + total_bytes_read, buffer, nbytes);
47              if (fwrite(buffer, 1, nbytes, fp) != nbytes) {
48                  logmsg(LOG_ERR, 1, "SSH download error - Error
    writing local file.\n");
49                  fclose(fp);
50                  sftp_close(file);
51                  ssh_disconnect(session);
52                  ssh_free(session);
```

```
53                  return -1;
54              }
55          } else if (nbytes < 0) {
56              logmsg(LOG_ERR, 1, "SSH download error - Error
    reading remote file.\n");
57              fclose(fp);
58              sftp_close(file);
59              ssh_disconnect(session);
60              ssh_free(session);
61              return -1;
62          }
63      } while (nbytes > 0);
64
65
66
67      // close local file and remote file
68      fclose(fp);
69      sftp_close(file);
70
71      // disconnect ssh session and free memory
72      ssh_disconnect(session);
73      ssh_free(session);
74
75    /* add ssh connection to attack record */
76      logmsg(LOG_DEBUG, 1, "SSH connection - Adding connection
    to attack record.\n");
77      int status = add_download("SSH", 6, r_addr, r_port, user,
    NULL, filename, remote_path, NULL, nbytes a);
78      if(status != 0){
79        logmsg(LOG_ERR, 1, "SSH download error - Error adding
    download to attack record.\n");
80      }
81      logmsg(LOG_NOTICE, 1, "SSH connection - %s attached to
    attack record.\n", save_file);
82
83      logmsg(LOG_NOTICE, 1, "SSH download - File succesfully
    download.\n");
84      return 0;
85  }
```

8.5.5 Function get_ssh_resource_by_scp

```
1  int get_ssh_resource_by_scp(const char* user, const char* host
       , const char* remote_path, const char* local_path, Attack*
       attack, ssh_session ssh){
2    scp_session scp = scp_new(ssh, SCP_READ, remote_path);
3    if (scp == NULL) {
4          logmsg(LOG_ERR, 1, "SSH download error - SFTP session
       cannot be created.\n");
```

```
5         ssh_disconnect(ssh);
6         ssh_free(ssh);
7         return -1;
8     }
9
10    status = scp_init(scp);  // inicializa la sesi n SCP
11    if (status != SSH_OK) {
12        logmsg(LOG_ERR, 1, "SSH download error - Error
    initializing SFTP session.\n");
13        scp_free(scp);
14        ssh_disconnect(ssh);
15        ssh_free(ssh);
16        return -1;
17    }
18
19    sftp_file file = scp_open(scp, remote_path, O_RDONLY, 0);
    // abre el archivo remoto en modo lectura
20    if (file == NULL) {
21        logmsg(LOG_ERR, 1, "SSH download error - Error opening
    remote file.\n");
22        scp_free(scp);
23        ssh_disconnect(ssh);
24        ssh_free(ssh);
25        return -1;
26    }
27
28  FILE* f = fopen(local_path, "wb");  // abre el archivo local
    en modo escritura binaria
29    if (f == NULL) {
30        logmsg(LOG_ERR, 1, "SSH download error - Error openng
    local file.\n");
31        scp_close(file);
32        scp_free(scp);
33        ssh_disconnect(ssh);
34        ssh_free(ssh);
35        return -1;
36    }
37
38    char buffer[1024];
39  char *data = NULL;
40    int  = 0, data-size = 0;
41  do {
42        nbytes = scp_read(file, buffer, sizeof(buffer));
43        if (nbytes > 0) {
44      data_size += nbytes;
45      data = realloc(data, data_size);
46      memcpy(data + total_bytes_read, buffer, nbytes);
47              if (fwrite(buffer, 1, nbytes, fp) != nbytes) {
48                  logmsg(LOG_ERR, 1, "SSH download error - Error
    writing local file.\n");
49                  fclose(fp);
50                  scp_close(file);
```

```
51                  ssh_disconnect(session);
52                  ssh_free(session);
53                  return -1;
54              }
55          } else if (nbytes < 0) {
56              logmsg(LOG_ERR, 1, "SSH download error - Error
    reading remote file.\n");
57              fclose(fp);
58              scp_close(file);
59              ssh_disconnect(session);
60              ssh_free(session);
61              return -1;
62          }
63      } while (nbytes > 0);
64
65
66
67      // close local file and remote file
68      fclose(fp);
69      scp_close(file);
70
71      // disconnect ssh session and free memory
72      ssh_disconnect(session);
73      ssh_free(session);
74
75    /* add ssh connection to attack record */
76      logmsg(LOG_DEBUG, 1, "SSS connection - Adding connection
    to attack record.\n");
77      int status = add_download("SSH", 6, r_addr, r_port, user,
    NULL, filename, remote_path, NULL, nbytes a);
78      if(status != 0){
79        logmsg(LOG_ERR, 1, "SSH download error - Error adding
    download to attack record.\n");
80      }
81      logmsg(LOG_NOTICE, 1, "SSH connection - %s attached to
    attack record.\n", save_file);
82
83      logmsg(LOG_NOTICE, 1, "SSH download - File succesfully
    download.\n");
84      return 0;
85  }
```

JSON simplification script

```
1  import json
2
3  # Ruta al archivo JSON de entrada
4  json_file = 'C:\\Users\\Lenovo\\Downloads\\attackers.json'
5
```

```
# Ruta al archivo JSON de salida
output_file = 'C:\\Users\\Lenovo\\Downloads\\remote_ips.json'

# Lista para almacenar los valores de remote_ip
remote_ips = []

# Leer el archivo JSON l nea por l nea
with open(json_file) as file:
    for line in file:
        try:
            # Cargar cada l nea del archivo JSON
            data = json.loads(line)

            # Extraer el valor del campo "remote_ip"
            remote_ip = data['attack_connection']['remote_ip']
            remote_ips.append(remote_ip)
        except json.JSONDecodeError:
            continue

# Crear un diccionario con la lista de remote_ips
output_data = {'remote_ips': remote_ips}

# Escribir el diccionario en el archivo JSON de salida
with open(output_file, 'w') as outfile:
    json.dump(output_data, outfile)

print("Los valores de remote_ip se han guardado en el archivo
    remote_ips.json.")
```

IP Address Geolocation Script

```
import json
import requests
import urllib3
urllib3.disable_warnings(urllib3.exceptions.
    InsecureRequestWarning)


# URL de la API GeoJS
url = "https://get.geojs.io/v1/ip/geo/{ip}.json"

# Cargar las direcciones IP desde el archivo remote_ips.json
with open("C:\\Users\\Lenovo\\Downloads\\remote_ips.json") as
    file:
  remote_ips = json.load(file)

# Diccionario para almacenar los datos de geolocalizaci n y
    frecuencia
geolocations = {
```

```
    "ip": {},
    "country": {},
    "country_code": {},
    "country_code3": {},
    "continent_code": {},
    "city": {},
    "region": {},
    "latitude": {},
    "longitude": {},
    "accuracy": {},
    "timezone": {},
    "organization_name": {},
    "asn": {},
    "organization": {}
}

# Iterar sobre cada direcci n IP y obtener la informaci n de geolocalizaci n
for ip_address in remote_ips:
    response = requests.get(url.format(ip=ip_address), verify=False)
    if response.status_code == 200:
        data = response.json()

        ip = data.get("ip", "")
        country = data.get("country", "")
        country_code = data.get("country_code", "")
        country_code3 = data.get("country_code3", "")
        continent_code = data.get("continent_code", "")
        city = data.get("city", "")
        region = data.get("region", "")
        latitude = data.get("latitude", "")
        longitude = data.get("longitude", "")
        accuracy = data.get("accuracy", 0)
        timezone = data.get("timezone", "")
        organization_name = data.get("organization_name", "")
        asn = data.get("asn", "")
        organization = data.get("organization", "")

        # Actualizar la frecuencia de cada valor en los campos correspondientes
        geolocations["ip"][ip] = geolocations["ip"].get(ip, 0) + 1
        geolocations["country"][country] = geolocations["country"].get(country, 0) + 1
        geolocations["country_code"][country_code] = geolocations["country_code"].get(country_code, 0) + 1
        geolocations["country_code3"][country_code3] = geolocations["country_code3"].get(country_code3, 0) + 1
        geolocations["continent_code"][continent_code] = geolocations["continent_code"].get(continent_code, 0) + 1
        geolocations["city"][city] = geolocations["city"].get(
```

```
    city, 0) + 1
60          geolocations["region"][region] = geolocations["region"
    ].get(region, 0) + 1
61          geolocations["latitude"][latitude] = geolocations["
    latitude"].get(latitude, 0) + 1
62          geolocations["longitude"][longitude] = geolocations["
    longitude"].get(longitude, 0) + 1
63          geolocations["accuracy"][accuracy] = geolocations["
    accuracy"].get(accuracy, 0) + 1
64          geolocations["timezone"][timezone] = geolocations["
    timezone"].get(timezone, 0) + 1
65          geolocations["organization_name"][organization_name] =
     geolocations["organization_name"].get(organization_name,
    0) + 1
66          geolocations["asn"][asn] = geolocations["asn"].get(asn
    , 0) + 1
67          geolocations["organization"][organization] =
    geolocations["organization"].get(organization, 0) + 1
68
69 # Guardar los datos de geolocalizaci n y frecuencia en el
    archivo geolocations.json
70 with open("geolocations.json", "w") as file:
71      json.dump(geolocations, file, indent=4)
```

Time difference calculation script

```
1 import json
2 from datetime import datetime
3
4 # Ruta al archivo JSON de entrada
5 json_file = 'C:\\Users\\Lenovo\\Downloads\\times.json'
6
7 # Ruta al archivo JSON de salida para las diferencias y la
    media
8 output_file = 'C:\\Users\\Lenovo\\Downloads\\differences.json'
9
10 # Lista para almacenar las diferencias entre end_time y
    start_time
11 differences = []
12
13 # Leer el archivo JSON de tiempos
14 with open(json_file) as file:
15      data = json.load(file)
16
17      # Calcular las diferencias y almacenarlas en la lista
    differences
18      for item in data:
19          start_time = datetime.fromisoformat(item['start_time'
    ])
```

```
20          end_time = datetime.fromisoformat(item['end_time'])
21          difference = (end_time - start_time).total_seconds()
22          differences.append(difference)
23
24  # Calcular la media de las diferencias
25  if differences:
26      mean = sum(differences) / len(differences)
27  else:
28      mean = 0
29
30  # Crear un diccionario con las diferencias y la media
31  output_data = {'differences': differences, 'mean': mean}
32
33  # Escribir el diccionario en el archivo JSON de salida
34  with open(output_file, 'w') as outfile:
35      json.dump(output_data, outfile)
36
37  print("Las diferencias y la media se han guardado en el
       archivo differences.json.")
```

Bibliography

[1] B. Santander. ¿qué es un honeypot y para qué sirven? [Online]. Available: https://www.bancosantander.es/glosario/honeypot

[2] N. Provos. A virtual honeypot framework. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/full_papers/provos/provos_html/honeyd.html

[3] I. S. N. C. Institute), "Industrial honeypot implementation guide," vol. 1, p. 45, October 2019.

[4] J. E. L. d. V. Eduardo Gallego, "Honeynet: Aprendiendo del atacante," vol. 1, p. 10, October 2019.

[5] F. J. M. Coll, "Ped: Red de equipos trampa de rediris," vol. 1, p. 10, October 2019.

[6] L. Spitzner, "Honeypots: Tracking hackers," *Addison Wesley*, vol. 1, p. 480, September 2002.

[7] DinoTools. Dionaea. [Online]. Available: https://dionaea.readthedocs.io/en/latest/introduction.html

[8] telekom security. T-pot. [Online]. Available: https://github.com/telekom-security/tpotce

[9] huuck. Adbhoney. [Online]. Available: https://github.com/huuck/ADBHoney

[10] CISCO. Cisco secure firewall asa. [Online]. Available: https://www.cisco.com/c/en/us/products/security/adaptive-security-appliance-asa-software/index.html

[11] CVE. Vulnerabilidad cve-2018-0101. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-0101

[12] N. Provos. Honeyd. [Online]. Available: https://www.honeyd.org/general/

[13] T. H. Project. Kippo. [Online]. Available: https://www.honeynet.org/projects/old/kippo/

[14] M. Oosterhof. Cowrie. [Online]. Available: https://cowrie.readthedocs.io/en/latest/README.html#what-is-cowrie

[15] zeroq. Amun. [Online]. Available: https://github.com/zeroq/amun

[16] mushorg. Glastopf. [Online]. Available: https://github.com/mushorg/glastopf

[17] buffer. Thug. [Online]. Available: https://thug-honeyclient.readthedocs.io/en/latest/intro.html

[18] armedpot. Honeytrap. [Online]. Available: https://github.com/armedpot/honeytrap

[19] mushorg. Conpot. [Online]. Available: https://conpot.readthedocs.io/en/latest/faq.html

[20] T. H. Project. Snare. [Online]. Available: https://www.honeynet.org/projects/active/snare-and-tanner/

[21] pwnlandia. Shockpot. [Online]. Available: https://github.com/pwnlandia/shockpot

[22] Cymmetria. Honeycomb. [Online]. Available: http://honeycomb.cymmetria.com/en/latest/cli.html

[23] evilsocket. Medusa. [Online]. Available: https://github.com/evilsocket/medusa

[24] T. H. . M. Eckert. Inetsim: Internet services simulation suite. [Online]. Available: https://www.inetsim.org/

[25] I. I. A. N. Authority. Protocol numbers. [Online]. Available: https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xml

[26] Virustotal. Virustotal. [Online]. Available: https://support.virustotal.com/hc/en-us/articles/115002126889-How-it-works

[27] Kaspersky. Troyanos. [Online]. Available: https://www.kaspersky.es/resource-center/threats/trojans

[28] I. I. N. de Ciberseguridad. Mirai. [Online]. Available: https://www.incibe.es/ciudadania/servicio-antibotnet/info/mirai

[29] A. W. Services. Introducción a aws. [Online]. Available: https://aws.amazon.com/es/getting-started/

[30] ——. What is amazon ec2? [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html

[31] ——. What is elastic load balancing? [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/what-is-load-balancing.html

[32] Microsoft. Visual studio code. [Online]. Available: https://www.bancosantander.es/glosario/honeypot

[33] ——. Why did we build visual studio code? [Online]. Available: https://code.visualstudio.com/docs/editor/whyvscode