Politecnico di Torino

Cybersecurity for Embedded Systems
01UDNOV

Master's Degree in Computer Engineering

Honeypot mechanisms for IoT clusters

Project Report

Candidates:
De Luca Antonio (s282133)
Guglielminetti Iacopo (s290289)
Manca Edward (s292493)
Silvano Francesca (s282021)

Referents:
Prof. Paolo Prinetto
Dr. Matteo Fornero
Dr. Vahid Eftekhari

Contents

List of Figures

Abstract

Honeypots are cybersecurity expedients that aim at luring attackers, exposing less or more realistic services. Attackers should not realize to be dealing with such decoy systems.
The taxonomy of honeypots is wide, many categories exist: Malware, DoS, Web Application are among them. Although being similar, they all serve for a different purpose.
This report gives an overview of the different honeypot flavors and the attacks they aim at constrasting.
The work also focuses on two use cases of two systems deploying honeypots against two different threats: a ransomware attack and a DoS attack.
Moreover, a close look to the implementation of both solutions is provided.
Finally, their results and final considerations are taken into account.

CHAPTER 1

Introduction

A Honeypot is a sacrificial system intended to attract cyber attackers, it mimics a target and gains information about cybercriminals and how they operate. This project aims at providing an overview of the state-of-the-art regards honeypots and describes two implementations of Honeypot inside a cluster IoT.
The document gives an overview of the various type of Honeypot to better understand the final goal of the project: develop a honeypot mechanism capable of attracting the attacker and giving feedback of what the attacker was trying to do. Two solutions have been presented for different types of clusters and attacks: one solution for a honeypot in a fully connected mesh cluster that protects from Malware attacks and a second one for a tree structure cluster that protects from DoS attacks.
The remainder of the document is organized as follows. In Chapter 2 it has been analyzed the various honeypot classification; in Chapter 3 it has been proposed a closer look at various honeypot typologies according to the attack; Chapter 4 it has been introduced an overview of the two solutions; in Chapter 5 it has been analyzed the implementation of the two solutions and in Chapter 6 It is possible to see the results.

CHAPTER 2

Background

2.1 Honeypot

A honeypot is an expedient that aims at attracting or tricking somebody or something. It has to be vulnerable and a realistic environment at the same time, resulting in an appealing decoy for attacks. A honeypot could be built for a specific purpose, as well as for more generic ones, it could be implemented by exploiting hardware and/or software and disposed of in different positions inside the network.
For what concert IoT systems, a honeypot represents a vulnerable environment that has to be targeted by hackers and then collect data about attacks, study their features and the tools used by the attackers. Many criteria are available to distinguish honeypots, some of which are listed below.

2.2 According to the level of interaction

Low-interaction honeypot: Honeypots belonging to this category have limited interaction with external systems.
There is no operating system for attackers to interact with, they represent targets to attract or attackers to detect by using software that emulates features of a particular operating system and network services on a host operating system.
The main advantage of this type of honeypot is that it is very easy to deploy and maintain and it does not involve any complex architecture.
On the other hand, it shows some drawbacks: it will not respond accurately to exploits. This drawback lowers the capability of detecting attacks.
Low-interactive honeypots are a safe and easy way to gather info about the frequently occurring attacks and their source. Some examples of low-interaction honeypots are listed below:

1. ADBHoney;
2. Heralding;
3. Honeypy;
4. HoneySAP;
5. Dionaea.

High-interaction honeypot: this is the most advanced honeypot class. These honeypots offer a very high level of interaction with the intrusive system. They give more realistic experience to attackers and gather more information about intended attacks; this also involves a very high risk to

catch the whole honeypot.
High-interaction honeypots are the most complex and time-consuming to design and manage. Also, they are very useful when we want to capture details of vulnerabilities or exploits the ones that are not yet known.

Medium-interaction honeypot: also known as mixed-interactive honeypots.
Medium-interaction honeypots are slightly more sophisticated than low-interaction honeypots but less sophisticated than high-interaction honeypots. They provide the attacker with an operating system so that complex attacks can be attracted and analyzed.

2.3 According to the purpose

Production honeypot: This honeypot typology is used as a defense instrument by an organization or on networks.
It could be deployed within the production network of the organization where services are placed and exposed.
These honeypots study and analyze attacks received and then help to strengthen the perimeter of the network. Such honeypots could also find and report some vulnerabilities of the environment used in the production network.
Research honeypot: Research honeypots are born with a purpose similar to production honeypots, but they have some different features.
Their main purpose is the improvement of defensive and prevention tools, with slightly different strategies concerning the production honeypots: the latter ones are more generic.
Research honeypots usually look for new types of malware and, moreover, these honeypots are the most used ones when redacting articles and papers for the cybersecurity community, to provide the best knowledge.
Research honeypots don't always simulate the environment of an organization network, they search for attacks faded to common infrastructure or solutions.
This type of honeypot could be classified into more categories: the **anti-spam honeypot**, which studies the strategies adopted by spammers, and the **malware honeypot**, which is more focused on the analysis of malicious software. In general, they are focused on strategies and techniques adopted by hackers for some specific types of attacks, on the vulnerabilities that these can exploit and on the malicious files that can be diffused.
All the information that can be collected about these attacks assumes a fundamental role and need to be saved in ad-hoc infrastructure. A detailed analysis of this data could allow the prevention of future attacks. For example, anti-virus industries use this kind of honeypot to update their database of attacks with the newly discovered malware.
Research honeypots have a more complex architecture. They are usually put outside of the organization network to avoid propagation of possible attacks and to gather all possible information since they are more exposed to attacks. The kinds of services that are exposed by these honeypot types require an accurate selection, the same for tools used and data stored.

2.4 According to the location

The honeypot could be placed:
Outside the network, before the firewall, in this case, the contact with the production network of the organization is null. This approach is used in research honeypots.
Inside the network: it emulates the real environments of the production network, and monitors possible attacks that act on the latter.
Here the visibility of attacks is complete, but the risk that the production network is attacked through

the honeypot itself needs to be taken into account.
Inside the demilitarized zone : DMZ is a network logically divided from the organization network. It has the purpose of offering some services to the public network. This solution is a compromise between the first and the second one.

2.5 According to the implementation

Physical honeypot: a real system with hardware and software connected through an IP address to the network;
Virtual honeypot: a software honeypot that simulates a configuration on hardware.

CHAPTER 3

Honeypot typologies for different attacks

In this section the research focuses on providing deeper details about specific attacks and classes of honeypots used to contrast them:

- Malware Attacks and Countermeasures;
- Web Application Honeypot;
- Virtual Honeypot;
- DoS Honeypot;
- MITM attack and honeypots;
- Eavesdropping Attacks and Countermeasures.

3.1 Malware Attacks and Countermeasures

3.1.1 Malware attacks

Malware attacks are common cyberattacks that exploit malicious software to execute unauthorized actions on the victim's system.
The burst of the internet contributed to a vast spread of various malware types. Among the most known types, there are:

- **Ransomware**, malware that threatens to publish the victim's personal data or block access to it unless a ransom is paid, encrypting data with malicious intent. Such attacks are usually carried out using a *trojan* disguised as a legitimate file (for instance, an attachment to an email) and relying on the poor social engineering skills of the users. However, this is not always the case, as the *WannaCry* ransomware proved, exploiting a vulnerability of the structure it spread across.
  The user immediately detects a ransomware attack due to payment warnings that pop up.
- **Spyware**, malware that is particularly hard to detect. It collects information about the victim's internet searches or even credit card data.
  Email attachments or downloads from file-sharing platforms represent typical injection methods for such malware.
  The victim usually experiences unusual behaviors of the system, such as new search engine tools.

3.1.2 Malware Honeypots

Defending yourself from malware attacks might not be trivial. Many people rely on antivirus software, that offers protection either for free or on a subscription.
Many antivirus firms rely on research honeypots to update their knowledge about the software that must be labeled as "dangerous" and create a *malware signature* that can be stored in a database of known threats.
You can use malware honeypots to update such databases, they are supposed to act as a decoy for attackers. In some cases, they should also incentive the attacker to infect a system, to study the behavior of the malicious software. This is the case of high interaction honeypots, that can even simulate entire systems to be more appealing to attackers.
On the other hand, you can also just use malware honeypots to defend yourself from attackers.
Literature offers multiple ways you can design and deploy malware honeypots, ranging from simple to complex solutions. Some of the studies are listed below:

1. The usage of sentinel files in a network against ransomware;
2. The monitoring of changes in files within a short interval of time;
3. The usage of software diversification;
4. The usage of machine learning algorithms.

Sentinel Files

A simple honeypot that intends to defend the system against ransomware, both the ones already known to the antivirus database and those not yet listed, could make use of sentinel files.
Such files act as *trip wires* for the trap to be triggered. You keep the original hash digest of the sentinel files aside. Since ransomware tends to encrypt data, if an attack is in progress you will see more and more files changing content. The strategy simply aims at monitoring changes in the digest of the sentinel files, to detect a ransomware attack and shutdown services before it spreads.
These files do not provide any production value and therefore every interaction with them (changes or removal) is to be considered malicious.
This is one of the few ways to deal with ransomware, since it is a particularly tough malware to defeat, it enters the victim's systems encrypted and so infection is nearly impossible to prevent.

Frequent changes to files

Some malware families aim at making the system busy by performing frequent changes to files within a short time. For small networks, you can monitor all file accesses and set a threshold above which you can state that probably a malware attack is in progress.
Heuristics can help since setting a low threshold might not allow authorized actions and a high threshold might not detect malware activities properly.
You can link thresholds of file changes to various degrees of response, as shown in the figure below.

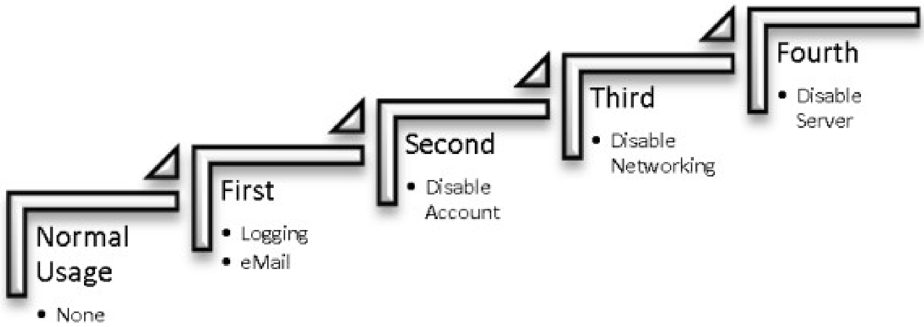

Figure 3.1: Gradual responses to file changes according to thresholds set by the admin.

This mechanism allows you to set your response gradually, avoiding the shutdown of all services at once and therefore avoiding having a large recovery time once the system is restarted.
A proprietary solution called *EventSentry* offers this file integrity monitoring service, letting the user deploy a honeypot that monitors files of interest via GUI.

Software Diversification

Studies prove that one of the keys to the success of malware attacks is that attackers rely on the so-called "software monoculture". In other words, it is much more likely that an attack is successful when the software implementation of the victim's system or infrastructure is easy to guess.
A *system call* is the way for a computer program to interact with the kernel of the operating system on which it is executed.
Two sets of system calls (well-known ones and hidden ones) can detect attacks if you let the trusted nodes only use the hidden ones, which are less likely to be guessed. Any traffic on the network that relies on the interface of well-known system calls is to be considered malicious.
This solution allows you to let malware run freely and without any risk to the assets you want to defend. Thus, you can study the behavior of the malware and gather more information about it.
A drawback of this approach is that the malware attack could utilize the code already present in allowed operations.

Machine Learning algorithms

Literature is full of machine learning-based approaches regarding malware detection. They all use some kind of malware classifier whose purpose is to distinguish between malicious and benign software, with a certain degree of confidence. Here the system exposes some fake services and learns how to defend itself thanks to a training period that exploits a training data set, called "EMBER" as described in the article "The Endgame Malware BEnchmark for Research" [4], which provides 900.000 training samples.
A classification technique often adopted in malware detection is the Decision Tree Algorithm, which classifies them based on malware features and behavior.
When malware is detected the honeypot gives some details to the admin, which is supposed to be a cybersecurity expert, who can validate or deny the classification, making the algorithm train even further.
As an example, ransomware always carries a message through which the user is warned that they should pay a ransom (often asked in untraceable currency, like bitcoins) to get their data back from the malicious encryption. This feature could be exploited for defense purposes because as soon as a malicious text is spotted through keywords such as "pay", "blocked", etc you stop its spread.

This is just an example since ransomware might not carry this message, or it might be encrypted. A high training period with a wide training data set could help you spot malware by features (e.g., some peculiar file size and code fragments) rather than by a ransom message.

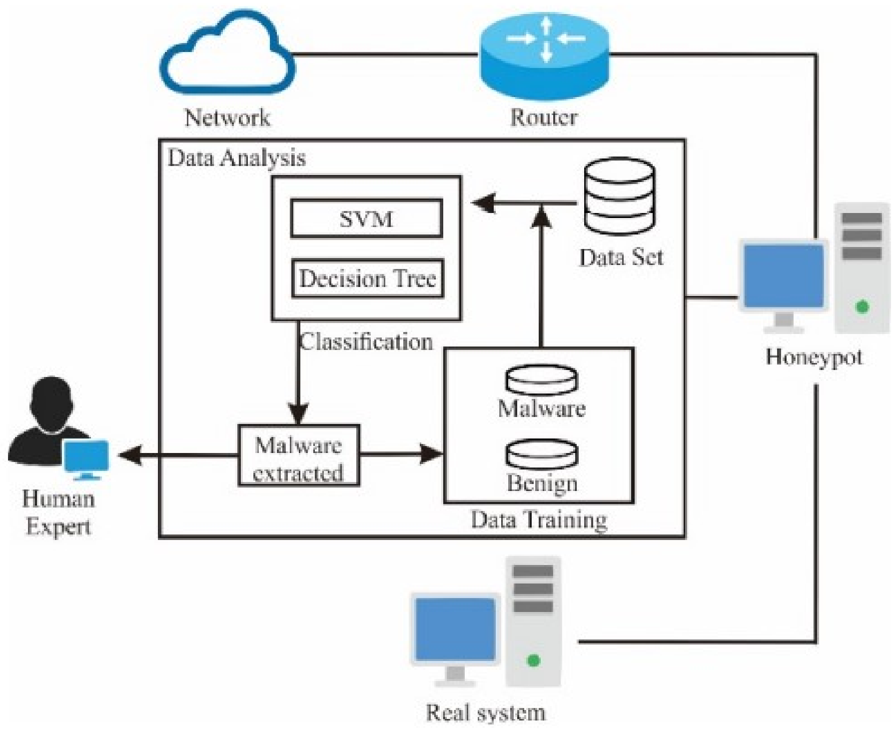

Figure 3.2: Example architecture of system with honeypot and machine learning.

3.2 Web application honeypot

End-user applications are an intrinsic part of an IoT infrastructure. These help users in monitoring and control IoT devices from remote locations.
These applications transmit user commands to the cloud, and subsequently, to IoT-enabled devices connected to the network.
Mobile apps, web apps, and desktop apps are the end-user applications found in an IoT infrastructure. Web applications often get attacked causing a break into confidentiality and integrity of information using:

- **SQL injection**: This is a type of attack granted by the fact that SQL doesn't check that whoever modifies the database has permission and a lack of input validation in the web application can cause the success of this attack. The attacker can insert structured Query Language code as parts of the final query string into a form that causes a web application to generate and send a query that works differently than the programmer intended;
- **Cross-site scripting (XSS)**: An XSS allows a cracker to insert or execute client-side code to collect, manipulate and redirect confidential information, or even view and modify data on servers or alter the dynamic behavior of web pages using any language (example JavaScript, VBScript, Flash);
- **Remote file inclusion attacks(RFI)**: This attack is caused when an application builds a path given by the user via variable without checking its origin. The attacker modifies a user variable in the URI (example GET POST in PHP) perhaps adding malign executable code, the web application downloads and runs the remote file;
- **Local file inclusion attacks(LFI)**: This attack is very similar to the previous one, in this case, the application only executes files included in the server so the attacker inside the script tricks the application by including another malignous script to execute.

In these cases, a honeypot can be very useful.
The Honeypot can individuate the IP source of the attack, the purpose of the attack (if there was the intention of destroying or modifying the database, etc), which methods are used by the attacker to capture data, and what is the most attacked part of the database.
The Web application Honeypot must respond to the attacker in the best possible way to better deceive him/her.

3.2.1 DShield Web Honeypot

The idea of this Honeypot starts with DSHield, a firewall log correlation system used by the SANS institute to collect logs received by volunteers worldwide and analyze them providing which IPs are more dangerous, which ports are more used for attacks, etc etc.
The honeypot is a low-latency one that collects logs from web apps adding them to all the logs that already are sent to DShield, this log contains the URL and header information such as IP address, host, user agent, referring from all requests (even harmless ones) and, after being checked with expressions in *config.txt* file, they are saved inside the logs folder of the honeypot and then posting it to the DShield database (this is done every 30 minutes).
The honeypot works simply: on the inside, it contains a set of templates and a login system; when an attack takes place, the template is chosen checking inside the set if a suitable one is present (if there is no template, it returns a default web page), the honeypot sends the response to the attacker meanwhile the request is logged and sent to the DShield database.

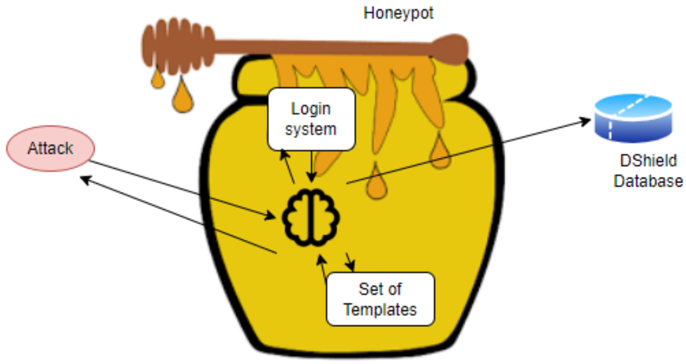

Figure 3.3: Structure of DShield honeypot

All of these ideas can be downloaded for free because all the logs are useful to the project. It is also used by home connections to collect data (as a peer-to-peer net "the more the better").

3.2.2 Glastopf

Glastopf is a very old Python web application low-interaction honeypot able to produce web applications vulnerable also to SQL injection. Glastopf manages to emulate vulnerability types, this allows us to manage multiple attacks of the same type until the attacker finds another fail in the web application or a new attack method.
Attackers will find the web service and try to attack the system, it is detected and gets logged by the honeypot that inserts all the information in the logs file (or database).
It has a good capability of logging based on the interaction that an attacker would have with the application but it shows some limitations: the front-end part is quite primary and attackers can easily recognize that this is not a real system and it only supports SQL injection, remote (RFI), and local file inclusion(LFI) vulnerabilities.

3.2.3 Comparation between glastopf and DShield Honeypot

In the paper of 2014 , *The Behavioural Study of Low Interaction Honeypots: DSHIELD and GLASTOPF in various Web Attacks*[6], the interaction of these two honeypots with various attacks is put in comparison.
DShield and Galastpof have similar working techniques.

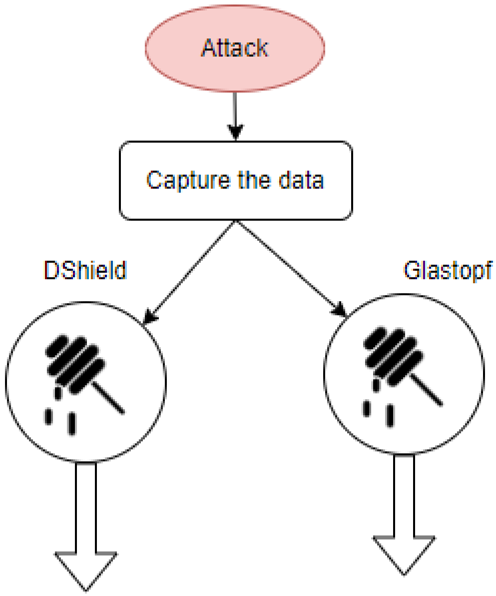

Figure 3.4: Flow of attack in DShield and Galastopf

DShield has a better ability to categorize the type of attack and utilizing the Apache server it is possible to extract a report. On the other hand, it is not easy to go through the logs and understand attacks easily.
Galastopf has logs that are easy to understand and they also capture the HTML information that is posted within the request, the response from the server and the response code (`200 OK`) confirming that the attack was successful on the other way cannot categorize the SQL injection and XSS and the logs generated for Remote File Inclusion attacks were lesser compared to Dshield.

3.2.4 SNARE and Tanner

SNARE (Super Next-generation Advanced Reactive honEypot) is a web application honeypot sensor attracting all sorts of maliciousness from the Internet.
It pays attention to the website, it has an inbuilt Cloner that, invoked before SNARE, clones all the web pages are given as input, all the images on the web pages, scripts and action elements so that the clone looks as good as a real system.
SNARE needs TANNER to work. It is a remote data analysis and classification service to evaluate HTTP requests and compose the response then served by SNARE. TANNER uses multiple application vulnerability type emulation techniques when providing responses for SNARE, so it creates each time a new session for a new attack, each session tries to detect the attacker (if it is a tool, a user, or a crawler), the location of the attack and shows through the TANNER UI for the administrator statistics regarding how many attacks of that kind have been found; after that, it emulates the response (via an emulator) and gives the attacker the response as the website would do.

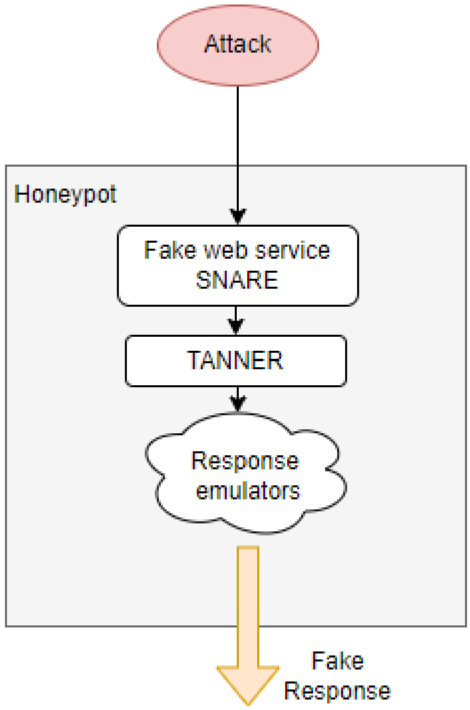

Figure 3.5: Structure of SNARE and TANNER

3.2.5 Solution of Honeypot with SNARE

In the paper *An Innovative Security Strategy using Reactive Web Application Honeypot*[8], a solution for a web application honeypot which includes many emulators for very complex vulnerabilities is proposed. Its structure is the following manner:

1. SNARE: It serves all the web pages on top of itself, becoming a server and hence monitoring all the HTTP events/flows throughout the application, similar to a real system;
2. TANNER: It analyzes the requests made through SNARE after that it generates the responses dynamically;
3. Database: for attack classifications;
4. Docker: Docker is used to create containers for the emulators; The emulator creates a container for different custom images, executes the payload of the attacker, takes the credible result, and deletes the container so that libraries are used inside a container and cannot damage the system, the honeypot remains secure;
5. Emulation engine: every event on SNARE has to pass through all types of emulators: GET, POST, and cookies emulators;
6. PHP Sandbox (PHPOX): it returns emulation results for emulators like PHP object/code injection, XXE injection, and RFI;
7. Base Emulator: it manages all the other emulators supporting multiple vulnerabilities and prepares the custom page for the injection;

8. Template Injection Emulator: this emulator imitates the Template Injection vulnerability. The input payload is matched with the element in the database to identify the type of attack, then it is injected into the docker vulnerable custom template of that type and, after the execution, the final results are sent to SNARE;

9. XML External Entity (XXE) Injection Emulator: the same story of the Template injection Emulator. It emulates XML External Entity Injection vulnerability and Out of Band XXE Injection as well;

10. PHP Object/Code Injection Emulator: it emulates the PHP Object Injection vulnerability. The injection results are sent to SNARE from the PHP sandbox so the code is executed in the PHP sandbox safely;

11. Attacker/Crawler Detection: it allows to detect if the attacker is a crawler or a tool and tries to get the owner.

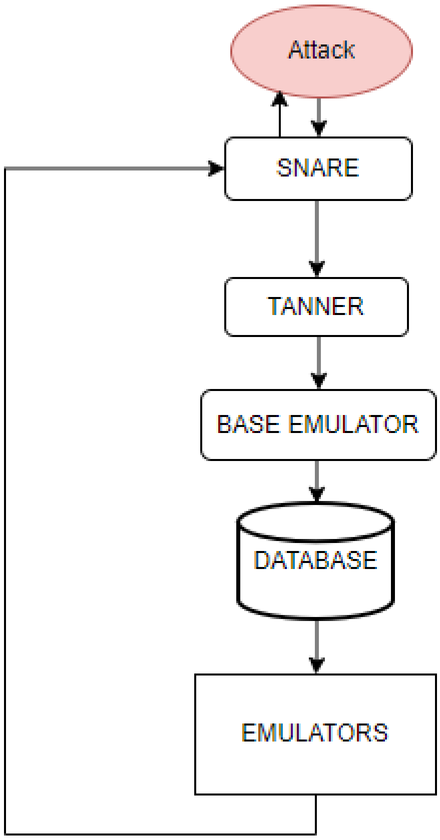

Figure 3.6: Structure of the honeypot

3.3 Virtual Honeypot

A virtual honeypot is a software that emulates a vulnerable system or network to attract intruders and study their behavior. It is easy to implement and unlike hardware-based, it can contain in a single

Virtual machine multiple honeypots implemented for different purposes and since it is all emulated it does not allow infiltrations from the outside that can use the same virtual honeypot to attack the system, but because it is completely emulated, a skilled hacker can see the difference from the real system.

3.3.1 A tool for virtual honeypot: Honeyd

Honeyd is a small daemon that creates virtual hosts on a network. It creates low-interaction honeypots that emulate the service of a real OS. It allows you to create multiple virtual honeypots (each with a different IP) by sharing the resources of the same server.
The server via TCP / IP protocol, based on the destination IP address, chooses which honeypot to serve.
It only supports TCP, UDP, and ICMP protocols.
Honeyd , as explained in the article [10] is designed for Unix systems and (as the other low-interaction honeypot) it has no OS installed, it has the advantage of running several different operating systems at the same time, each IP has his port to make the service listen and return different fake message returning them to the hacker, this fake message is generated by personality engine to make it look good and logical according to the template.
The template is a virtual machine where we can set which ports are open, which OS is running etc. each port can be set to be open with a script running on it to simulate the service.

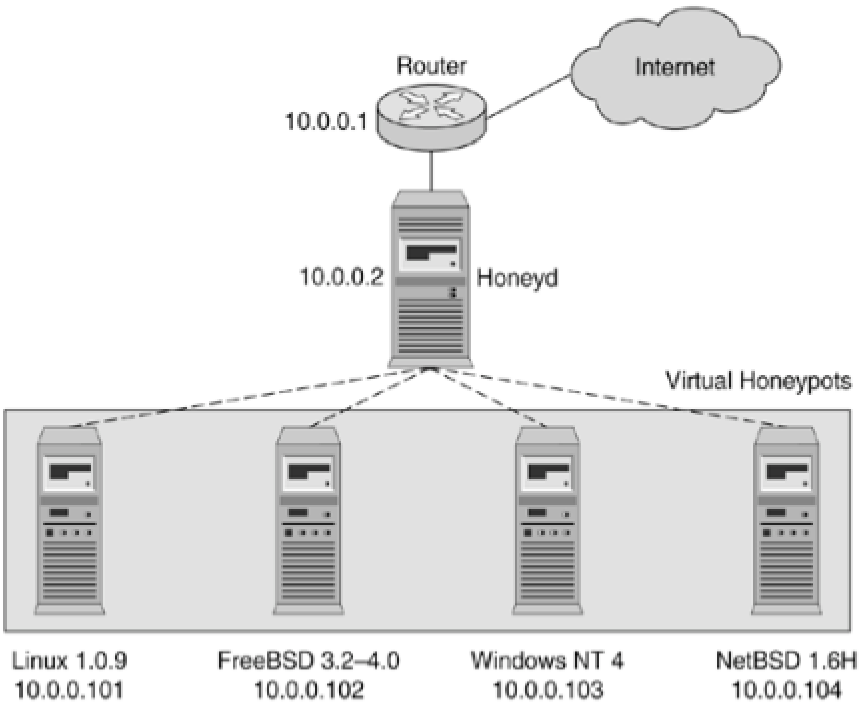

Figure 3.7: Structure of the honeyd

The file Honeyd.conf contains the configuration for the virtual network .
Once the template is fixed we can set the IP addresses, at the end we create a network that seems

real to the attacker.

As soon as a packet arrives, the dispatcher checks that the IP is valid in the configuration file, in the absence of the IP a default model is sent back.

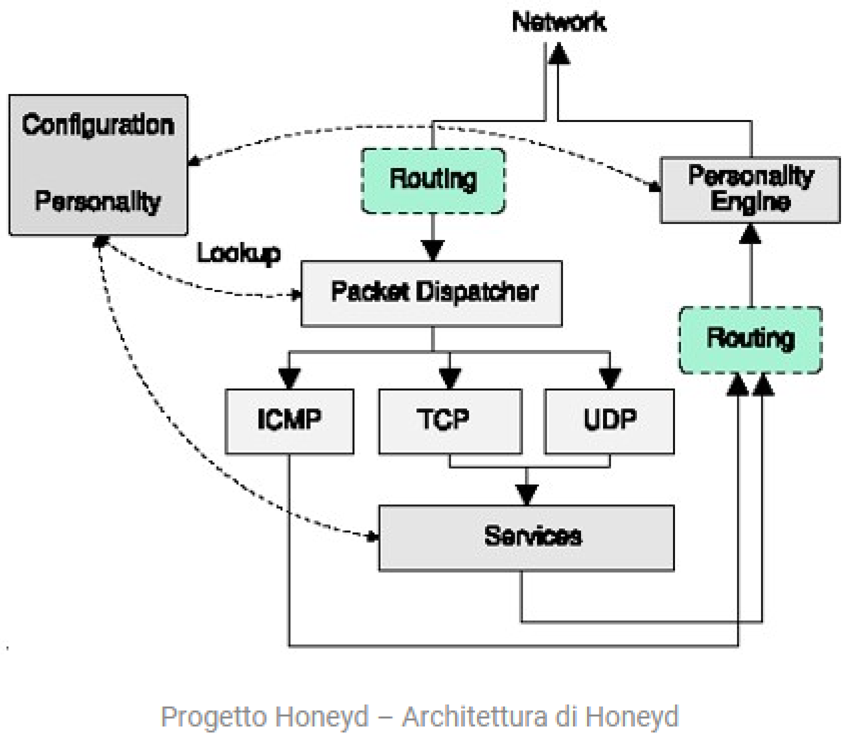

Figure 3.8: Implementation of honeyd

Upon receipt of a request, the framework checks whether the packet concerns an already established connection and in this case forwards it to the corresponding service, otherwise a new process is created for its execution. In addition to local services, a connection redirection mechanism is supported through which it is possible to route a service request to a real system. The personality engine and configuration engine work together to decide the protocol for transferring according to the configuration (transport and link layer), so before returning the packets to the network, the personality engine modifies the headers to make them compatible with the stack used in the machine's SO.
Since attackers often use tools to scan the network and know the operating system running on the system, by default Honeyd uses Nmap's fingerprinting database as a reference for TCP headers and Xprobe's for ICMP headers. In this way, he tries to deceive the attackers with their weapons. Since both the DOS Honeypot and Malware Honeypot works with TCP/IP protocol, we think it should be a good idea to run this daemon inside one Raspberry Pi and generate both the honeypots in virtual. Honeyd is a very interesting low interaction honeypot but its problem is its age, apparently, the project is outdated and nobody seems to upgrade it.
Another problem given by age is that hacker tools are evolving over time so identifying this honeypot is not so hard.
Now with a basic scan, it is possible to find which ports are open, connect to them and as soon as the connection is established the hacker (based on the response the honeyd provides) understands quickly that something is wrong and aborts the attack.

3.4 DoS attack

A DoS attack takes place when a client is used by the attacker to flood with TCP and UDP packets one target server overriding the resources of a system so that it cannot respond to service requests of legitimate users. This causes bandwidth saturation and the exhaustion of computing resources, even leading to a server crash.
A DDoS attack takes place when multiple systems target a single system with a DoS attack, more difficult to be tracked because the attack is launched by various positions.
Citing an example, one major use of IoT is home automation systems. If an attacker attacks the main server of such a system with a DoS attack, the whole system tends to shut down and any appliance within the whole house (sometimes even door locks) is made inaccessible. This small example shows the significance of security implementation in IoT. In IoT-based networks, new devices that enter the network are configured automatically due to their open nature. This leaves such networks prone to a lot of attacks. The open nature of IoT makes it relatively easy for spammers and attackers to infiltrate IoT networks and launch DoS attacks.

3.4.1 Different type of Dos attack

Unfortunately exist very different types of Dos attacks, based on the protocol that is used, if the aim is crash services or flood services. For example in a yo-yo attack, the attacker generates a flood of traffic until a cloud-hosted service scales outwards to handle the increase of traffic, then halts the attack, leaving the victim with over-provisioned resources. When the victim scales back down, the attack resumes, causing resources to scale back up again. An example of a TCP attack is **SYN Flood Attack**. To better understand it we need a closer look at how TCP initializes a connection between a client and a server.

The algorithm used by TCP to establish and terminate a connection is called a three-way handshake. The idea is that two parties want to agree on a set of parameters, which, in the case of opening a TCP connection, are the starting sequence numbers the two sides plan to use for their respective byte streams. In general, the parameters might be any facts that each side wants the other to know about. First, the client (the active participant) sends a segment to the server (the passive participant) stating the initial sequence number it plans to use (Flags = SYN, SequenceNum = x). The server then responds with a single segment that both acknowledges the client's sequence number (Flags = ACK, Ack = x + 1) and states its own beginning sequence number (Flags = SYN, SequenceNum = y). That is, both the SYN and ACK bits are set in the Flags field of this second message. Finally, the client responds with a third segment that acknowledges the server's sequence number (Flags = ACK, Ack = y + 1). The reason why each side acknowledges a sequence number that is one larger than the one sent is that the Acknowledgment field actually identifies the "next sequence number expected," thereby implicitly acknowledging all earlier sequence numbers. To better understand the step to establish the connection, please refer to the figure below.

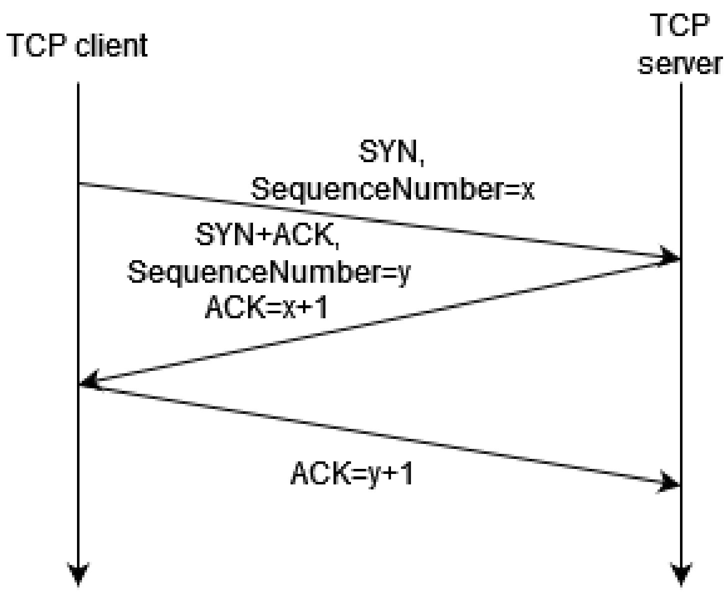

Figure 3.9: TCP handshake protocol

An attacker that wants to establish a **SYN Flood Attack** creates a bot that creates thousands of first segments in the 3-way handshake protocol. Then when the server replies with the SYN-ACK packet, the bot client never reply to it to make thousands of incomplete request of connection to the server to slow down it. Please refer to the picture below.

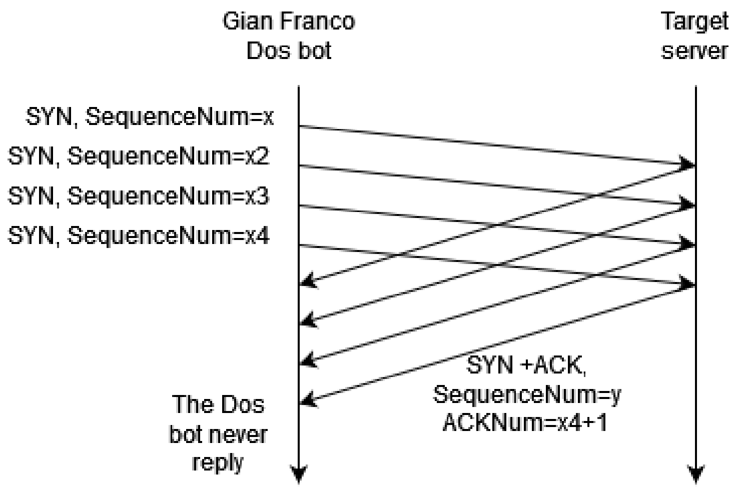

Figure 3.10: TCP SYN flood attack

Now, the following question is, could we use honeypots to avoid this type of TCP attack? According to what we found on the internet, to protect our servers from the **SYN Flood Attack**, is better to:

- Installing an IPS to detect anomalous traffic patterns;
- configure the onsite firewall for SYN Attack Thresholds and SYN Flood protection;
- Installing up-to-date networking equipment that has rate-limiting capabilities.

So honeypots are not useful to prevent that.

3.5 MITM attack and honeypots

In a server-client architecture, a MITM attack takes place when the attacker is inserted inside the communication between client and server. Let's make an example of possible situation that could occur, and a proposal of honeypot that could resolve it.
The normal procedure to avoid MITM attack is to encrypt and decrypt data between client and server. The main steps are:

1. **Use asymmetrical encryption**. First we use an asymmetric algorithm to allow us to send in a secure way the symmetric key that will be used in the future;

2. **Use symmetric encryption**. Now that the symmetric key is shard between the server and the client, the data are sent normally between the ends points and not anyone in the middle can understand them.

What if the first step fails? As an example,our IoT server for example could use a weak RSA algorithm, based on 16 bit key. Our men in the middle could manage to find the private key of the server and decrypt the symmetric key that will be used in the second step, like shown in the figure below.

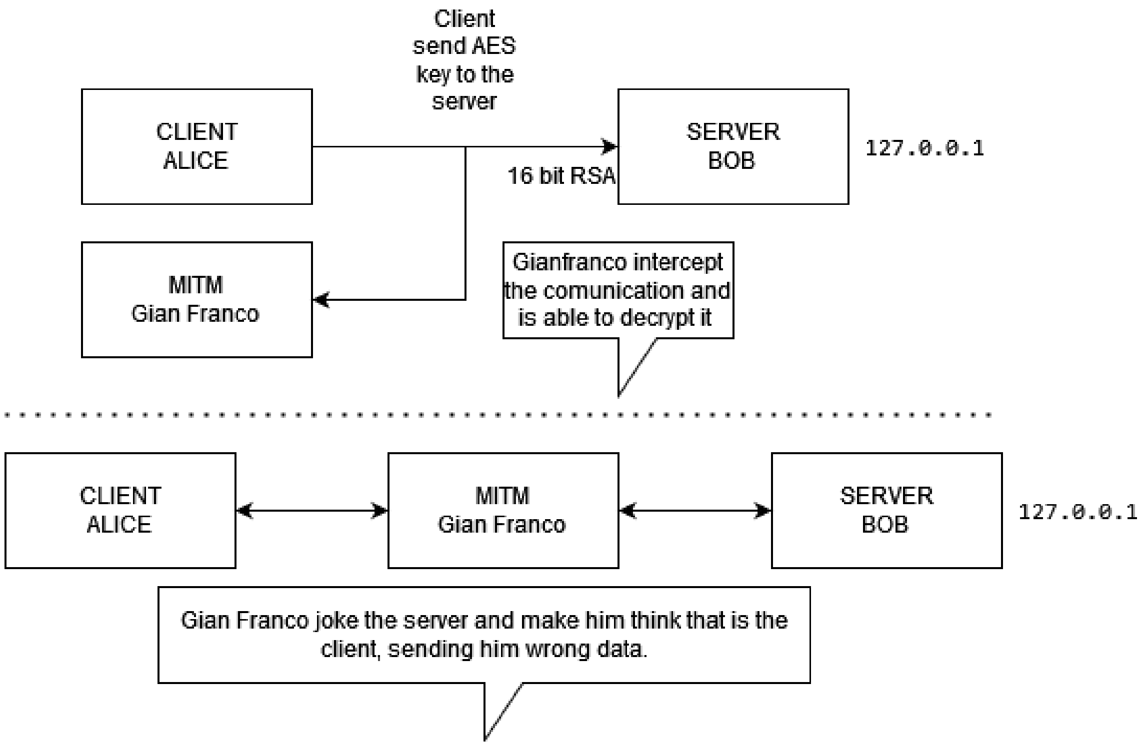

Figure 3.11: An example on MITM attack

How can our system understand that it is speaking to a MITM and not to the client? Our honeypot would intervene right here.
For example, it could first check if the wrong data sent by the MITM makes sense or not. Then it could check if the data of the fake sensor makes sense with respect to a set of data collected from the real sensor in the precedent days. If the honeypot recognizes that a man in the middle is present, it could collect data about the hacker and see what information they are trying to capture, which attacks he/she is trying to inject in to the network, studying the methodology used by hackers. Moreover, a honeypot AP could (at least temporarily) keep the hacker engaged and alert the administrator so

that the actual network can be safeguarded.then close the collection. If no violation is detected, the honeypot sends the data to the real server. A possible implementation is show in the figure below.

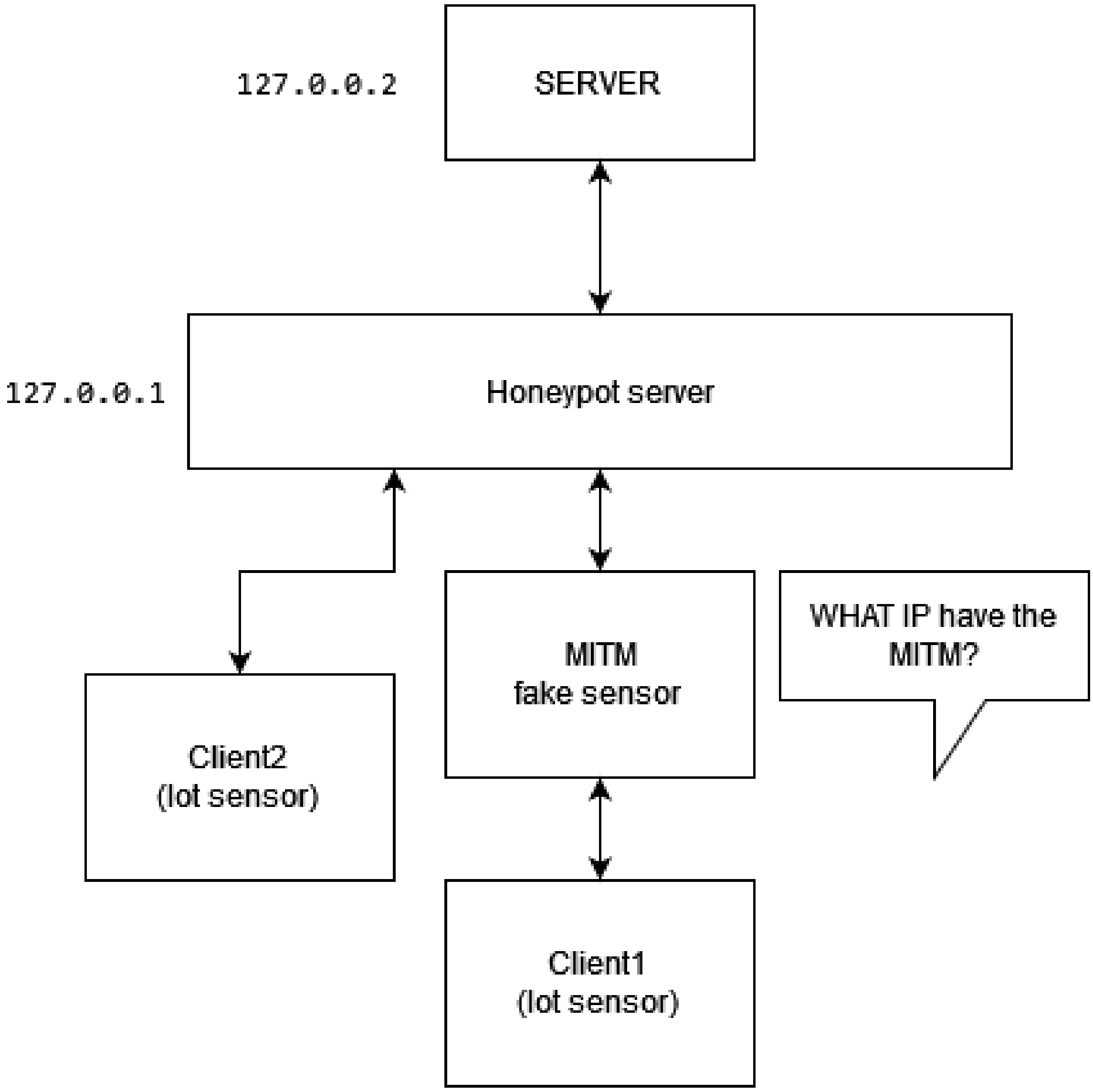

Figure 3.12: A possible system structure for the MITM honeypot.

3.6 FAKEHONEYMITM (MITM attacks and honeypots)

Regarding the MITM attack, we have another possible idea to implement. As we know, a man in the middle tries to place himself between the sensors of the IoT cluster and the servers inside the network. He need to intercept the message from the sensors to the servers and viceversa. But what if the sensor is not real but is a honeypot? For example our cluster could expose a connection between 2 honeypots, the first one is a fake server, the second one is a fake sensor. But why an attacker have to choose this connection for him malicious purpose? Well, this link could work with a weak encryption to seems like a vulnerability of our network. When the MITM manage to put itself between the fake connection, our honeypots could collect data on the activity of the hacker. To better understand this idea, please refer to the figure (3.13)

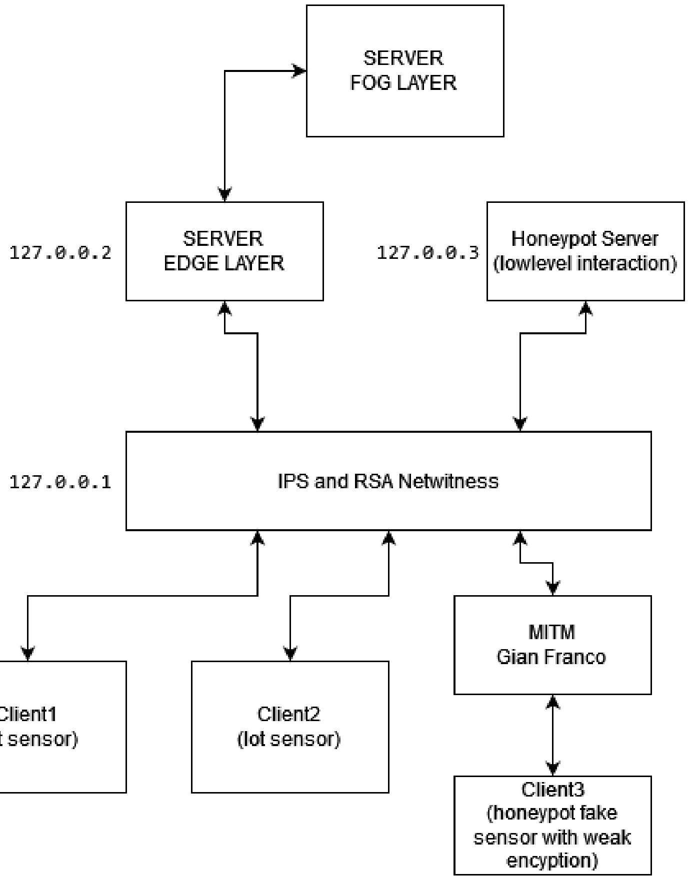

Figure 3.13: A possible structure for the fake sensor honeypot and MITM

3.7 Eavesdropping Attack and Countermeasures

3.7.1 Eavesdropping Attack

In eavesdrop attack the network is sniffed in order to retrieve information transmitted through the network itself; the process can be classified as passive or active depending on the existence of an interaction of the attacker on the network. Specifically, a passive attack is performed just analyzing packets of information on the eavesdropped network channel, while an active attack is performed requesting directly to the channel information the attacker wants to retrieve.
To counteract eavesdropping phenomena on a network several possibilities are present, always keeping in mind that the desired objective is to maintain secrecy of transmitted information.

3.7.2 Encrypting the channel

The most straightforward method to secure a channel communication is to encrypt the transmitted data, in this way even if the channel is sniffed data are secured (assuming a safe key maintenance and management). This solution is great for networks/clusters of powerful computers or for application that require not sending too many data, otherwise the risk is to spent most of the time computing cryptographic algorithms instead of computing for the cluster purpose. Mitigations could be the design of specific hardware to accelerate those computations; this implies that the applicability of this solution is impossible for most of the cheap low-cost low-power IoT-cluster applications.

3.7.3 Physical Countermeasures

Intelligent Reflecting Surface

A solution in proposed in the article *"Eavesdropping with Intelligent Reflective Surfaces: Threats and Defense Strategies"*[9] will enhance the point-to-point communication between the sender and the receiver making it impossible for the attacker to read the channel. This result is obtained exploiting properties of reflecting surfaces able to dynamically re-adapt and focus the connection to one specific point. Specifically, the channel for the attacker is worsen to a level of binary gibberish securing the information in the meanwhile.

Channel Capacity and Information Freshness

Other techniques are related to evaluating the capacity of the two receiving nodes (the actual receiver and the eavesdropper), CD and CE, and trying to set a network transmission protocol to make the eavesdropper unable to receive the data up to a given secrecy level RS as perfectly desctibed in the paper *"Relay Selection for Wireless Communications Against Eavesdropping: A Security-Reliability Tradeoff Perspective"* [11]
For systems in which the useful information is related to the freshness of the obtained data and the only secrecy that matters is the one aimed in protecting the newest data (we can imagine systems analyzing the position of cars on the road) other metrics have been developed ("secrecy age" and "secrecy age outage") as the article *"Secure Status Updates under Eavesdropping: Age of Information-based Physical Layer Security Metrics"*[7].

3.7.4 Impulsive Statistical Fingerprinting

This method is trying to move in the direction of cryptographic methods, without the usage of standard cryptographic algorithms, The point of this technique is to remove information that make the data understandable before transmitting it. The method relies on the exchange of fingerprints (e.g. time-stamps) between the source node and the destination node. These fingerprints are then used to compute statistics using some algorithms (final results will be mean, std deviation, other order momentum, and so on..) and to manipulate data before transmitting those by manipulations like removing the mean, whiten the data by dividing it by the std-deviation, etc. This method is suitable for the usage on low-performance IoT-clusters; on the other hand it is not mathematically secure: the method relies on the fact that an attacker will not be able to get a sufficient number of fingerprints to then be able to retrieve information on the transmitted data.

3.7.5 Active Eavesdropping - ML approach

All of previous remedies are for passive eavesdropping. For active eavesdropping there is the necessity to treat requests through the channel. Common possibility is to reduce the permission of requests from class of nodes. Another approach is to exploit ML to perform anomaly-detection classifying series of actions as normal or anomaly. Algorithms suitable for IoT-class devices can be random trees (random forests in the case of parallel execution on all the IoT-cluster), or comparison based like K-nearest neighbors, or linear methods like logistic regression or Perceptron based approach; in the case of NN (Neural Network) this must be conceived specifically for the device it will be deployed considering HW resources and computation capabilities. In all these cases features can be collected from the normal network traffic in a protected environment and then used to in the training procedure.

3.7.6 Possible Honeypots

Honeypots can work together with the identification procedure provided by previous techniques; as an example an useful honeypot approach could be in jamming the channel once the attacker is identified. Or keeping an unprotected (less protected) channel the attacker will try to get access to first and that is sending fake, but coherent, data.

3.8 Password Attack and countermeasures

3.8.1 List of counteracts and considerations

- Limit the number of attempts and track the requester;
- Since an attacker will try first to use common passwords maybe related to personal accessible info, a possibility is to keep a blacklist of possible passwords created from those personal info (that we also know..) and block the attacker attempts immediately;
- A honeypot approach could be, in the case of previous condition, to give access to a fake shell in order to evaluate requests from the attacker, while taking countermeasures.

CHAPTER 4

Implementation Overview

In this chapter we give a summary of the two very different solutions, thanks to the research done we realized that there are different types of clusters for IoT systems, so we decided to create two completely different solutions from each other, in order to be able to range across multiple IoT systems.

4.1 Case of study 1: Cluster and Honeypot for a Ransomware attack

4.1.1 Problem

Malware are one of the major threats nowadays. They continue evolving, making it difficult for countermeasures to be always up to date.
Among malware, particular importance is given to ransomware. Ransomware is a category of malware whose goal is to encrypt, maliciously, data on a target device. Ransomware creators aim at receiving a payment in order to provide the original data back to the owner. Studies show that even if the victim accepts to pay, not always all data is retrieved[3]. You should always consider that you are dealing with cyber criminals that don't have good intentions. The payment is often in untracked currencies, such as bitcoins.
Moreover, ransomware is a rising threat. The 2,084 ransomware complaints received by the IC3 in the first half of 2021 amounted to over $16.8 million in losses.[2][1]

4.1.2 Goal of the project

The goal of the project is to provide a possible countermeasure to ransomware attacks based on the sole usage of tripwire files ("sentinel files") spread in the file system and redirection of some linux commands, such as "sort -R".

4.1.3 Solution Overview

Most IoT solutions include devices that collect data from the environment and send it to more powerful components that gather such information, possibly perform computation and forward data elsewhere. Let's suppose to have a distibuted system as in the following picture.

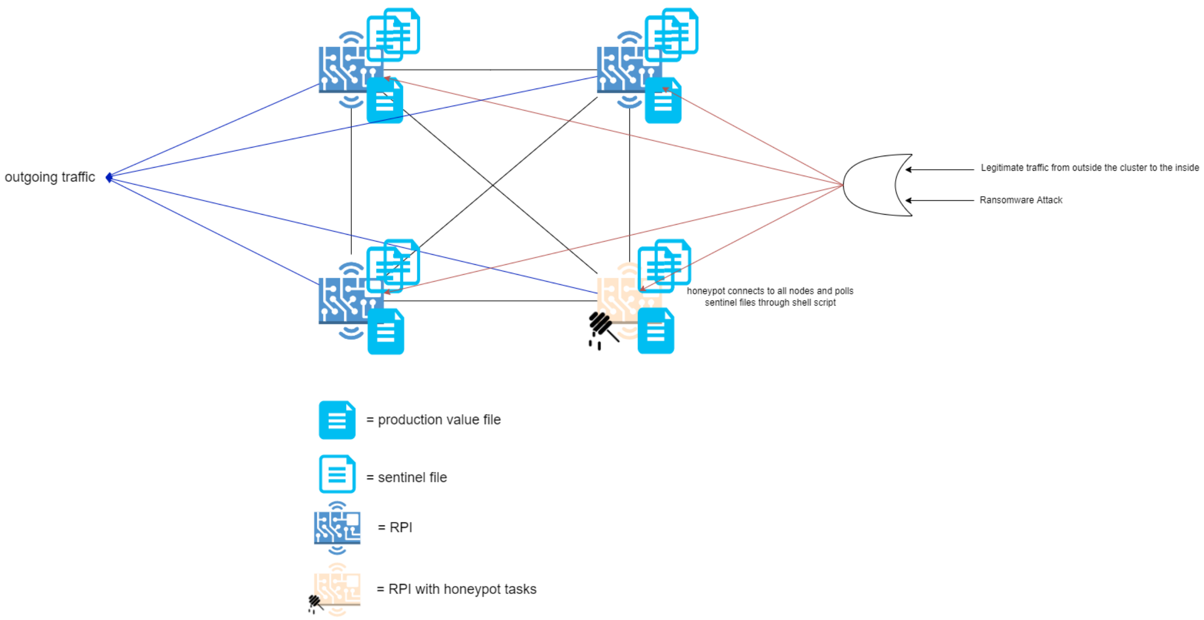

Figure 4.1: Example scenario of honypot against ramsonware attacks.

Cluster

Any Raspberry Pi in the picture represents a server, each connected to more clients (e.g., sensor nodes, not shown in the picture). Such servers are then connected among each other through a network and a communication protocol.
In this example, the architecture of the network is a full mesh, implying that each server is directly connected to any other. You could think of this network as either a local area network (LAN) or wide area network (WAN). They offer two different flavors of the same problem since in LANs you assume that all devices belonging to the network own a different local IP address and you should deal with the NAT system, while in a WAN servers have different public IP addresses and you should deal with DNS in order to communicate with each node in the network.
Raspberry devices exchange messages via MQTT, a lightweight publish/subscribe protocol.
Each device can manage messages of two kinds:

- Dispatched commands. They should arrive from the dispatcher in the network (not shown here). The ransomware will pretend to be the dispatcher to let a victim device execute malicious encryption;
- Attack notifications. If under attack, a device broadcasts a message stating that is the victim of an attack in progress.

Defensive Mechanism

Each raspberry stores both files that are needed for the various operations it has to perform (e.g., files storing data from sensors) and files that are useless to the purpose of the cluster. The latter ones are better known as "sentinel files". Nobody should ever interact with sentinel files since they are useless for benign purposes, thus every modification to them must be considered as malicious.
Each raspberry runs a periodic check on the status of its own sentinel files, by checking their hash digest. If a mismatch takes place, then an attack notification is broadcasted and the node performs a shutdown, both avoiding to encrypt all its file system and avoiding to infect other nodes in the cluster. This operation is crucial, it grantes that the ransomware cannot spread out of the victim's file system.

The infected raspberry is inserted in a blacklist by all other nodes in the cluster, its messages are not read anymore because they are considered malicious. The node is deleted from the blacklist after a fixed amount of time. During this interval, a technician is supposed to perform a reset of the infected device.
Masking (or redirection) of some linux commands is provided to help sentinel files always appear as first encountered by the ransomware. Masking is provided for some options of commands "ls", "sort" and "rm" that could affect the order thanks to which files are displayed in a directory.

Ransomware

Most real ransomware would have been too harmful and many of them could have put our systems in danger. We chose to build our own ransomware that claims to be the dispatcher and sends messages (containing bash commands) to a victim in the cluster.
Ransomware aim at encrypting the victim's file system thanks to asymmetryc encryption. They try to encrypt the victim's file system with their own public key, obliging the victim to restore data by applying decryption with the ransomware's private key, that is secret until a payment is received. Our ransomware performs the following steps:

1. Creation of a pair of asymmetric keys;
2. Injection of the public key in the victim's file system;
3. Encryption, directory by directory, of each file in the victim's file system using the public key.

4.2 Case of study 2: Cluster and Honeypot for a DoS attack

For this solution we took a cue from the paper *Use of Honeypots for Mitigating DoS Attacks targeted on IoT Networks*[5] where we found an analysis of the usage of a honeypot to reduce the damage of a DOS attack. One of the several possible attacks on IoT systems, Denial of Service (DoS) attack has been a nightmare for communication networks over the years and they pose a major security threat to IoT systems as well.

4.2.1 Goal of the project

The goal in this part of the project is to structure a tree cluster and elaborate an honeypot able to protect it from a DOS attack focusing on the IoT aspect of the project and make the solution really usable by these small devices.

4.2.2 Solution overview: DOSPOTPY and climatOffice

In the last years it becomes more and more important to avoid waste of energy at home and at work, due to the increasing cost of energy and to the environmental impact from the chain of the energy supply. So a startup develop a new IoT system named climatOffice(This is only an imaginary company, it doesn't exist in the real life). This solution aims at control the temperature in every office of the company, using N temperature sensors in every room an at least one air conditioner. The desired temperature could be set by an external terminal to the server that control every room. If there are windows in the room, the system also offer the possibility to control the illumination using automatics roll-up shutter. The last feature of this IoT cluster is to possibly control the accesses to some vulnerable rooms, like the server's farm room. So it can lock or unlock some doors. It can do this last thing automatically or by a command from a terminal. A possible Dos attack could for example avoid the door to close or open, or shut down the air conditioner system. To avoid so we could implement our systemr like in the figures below.

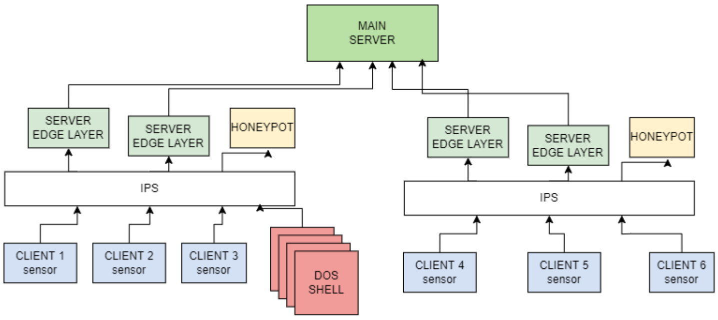

Figure 4.2: An example of the climatOffice IoT cluster

The cluster consists of a main server to which the various edge servers are connected so a client from the outside will turn out to be the leaf of this tree that can also evolve in multiple layers, in this case we preferred to remain fairly simple. The dispatcher, once the client sends a packet, redirects it to the server it is interacting with, every time a new connection arrives the dispatcher evaluates it as "harmful" then sends all the packets to the honeypot which, while the dispatcher continues the his job, checks that this client is credible and redirects the response to the dispatcher so that he is free to create a new connection with one of the servers. When a client is malicious (it sends too many packets, or with an unconfirmed structure) the honeypot writes the IP to a blacklist and then the dispatcher silences the connection, so the DOS attacker will seem to be sending data but actually can't slow down the system. In a litle system we could think of one dispatcher that manage all the servers indeed for a more larger one ,to better deliver the traffic and have less delay, is a good choice to have more than one dispatcher each one that manage a part of the cluster.
For example, our attacker implement a bot that connect to the system and create thousands of fake temperature sensor that send random and wrong data to the server. Our dispatcher usually send new traffic connection to the honeypot, that will analyse the data . If this is no sense data, the honeypot will store in a log file(black list) the IP addresses of the fake sensor and will told to the dispatcher to avoid accept new packets from that IP. In a more simple way of cluste rwe can think of having much fewer packets and therefore of having less traffic in the dispatcher, if so we can then merge the dispatcher and the honeypot so that the check is faster without fill the dispatcher too much.

4.2.3 The dispatcher evolution: IPS issue

In order to build our cluster we have starting from the structures gives by the paper that is shown below:

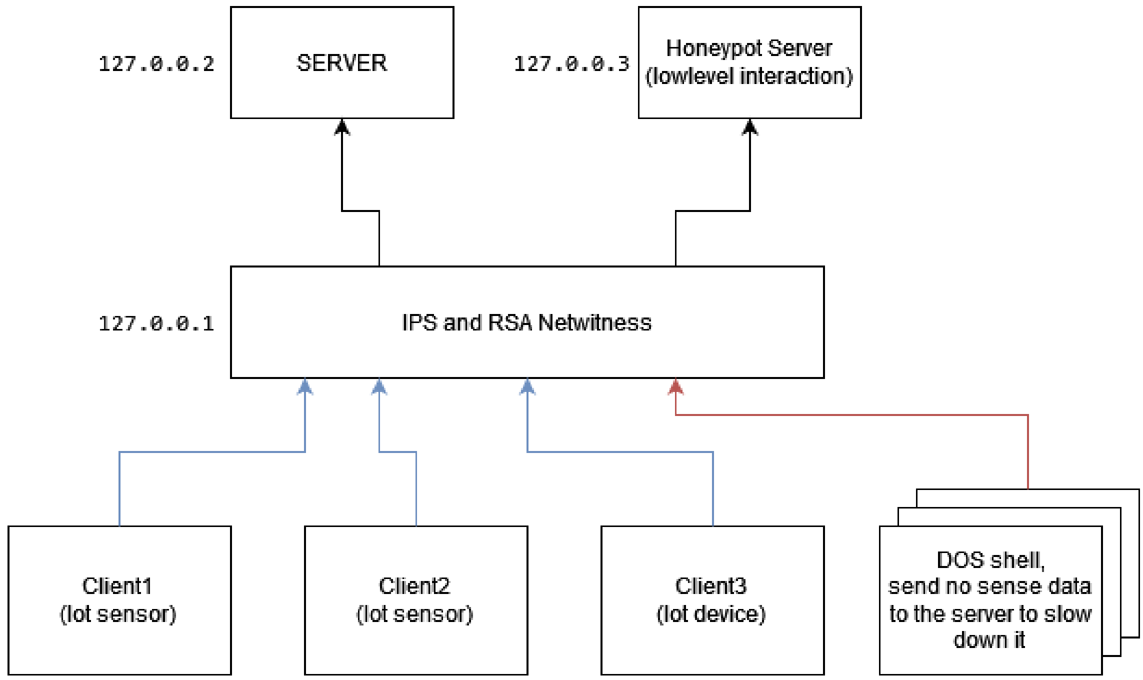

Figure 4.3: IoT network scheme with IPS.

In this first draft, which was then elaborated to arrive at the solution used in the implementation, we pay particular attention to the Intrusion Prevention System (IPS) that is an active protection system. It attempts to identify potential threats based upon monitoring features of a protected host or network and can use signature, anomaly, or hybrid detection methods. Unlike an Intrusion Detection System, an IPS takes action to block or remediate an identified threat. While an IPS may raise an alert, it also helps to prevent the intrusion from occurring. An IDS is a passive monitoring solution for detecting cybersecurity threats to an organization. If a potential intrusion is detected, the IDS generates an alert that notifies security personnel to investigate the incident and take remedies against the action.
In our very first proposal the system is less complex, because the IPS itself takes the decision on where to redirect the traffic to. If it is a malicious packet, to the honeypot that will collect data about it, if it is a normal one, to the server. The IPS needs time to take decisions, so the traffic could be slowed down by it. If the application doesn't require high availability, this solution is acceptable. Our second idea is reported in figure below.

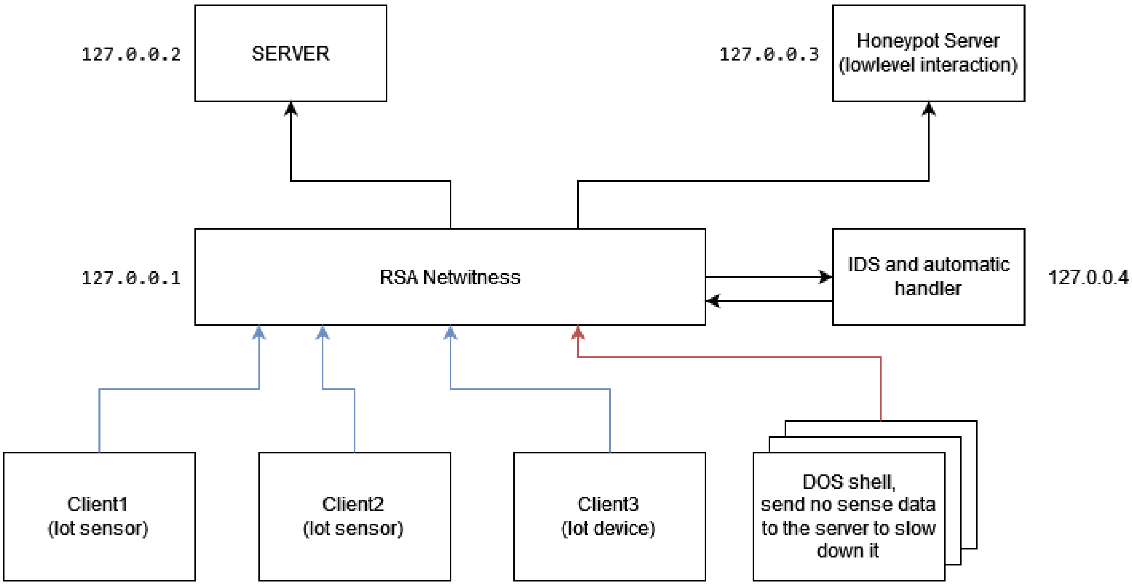

Figure 4.4: IoT network scheme with IDS.

This time an ad-hoc server has an IDS, that arises an alarm when something wrong is happening. Then an algorithm inside the server takes a decision on what to do with this possible malicious connection, and sends a command to the RSA server to redirect the traffic. In this way, the system has an high availability because the RSA server is not slow down by the IDS, that acts in a totally autonomous way. However, this idea requires more engineering effort and hardware so we redirect the solution to a more simple IPS to create a more realistic IoT cluster like in the figure below.

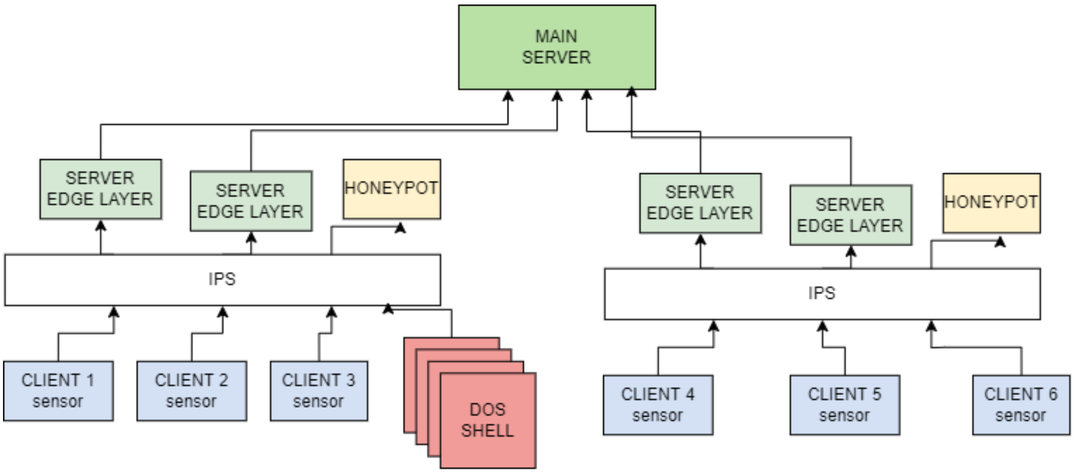

Figure 4.5: IOT network scheme level 2 with dispatcher

In this solution all the traffic generated by the clients is redirected to the various servers that the IPS serves, when an attack occurs It knows it by checking if it is a new IP or if the structure of the packets is wrong and redirect the traffic to the honeypot. In a second look we noticed that IPS was simple and much more like a dispatcher redirecting packets, thats why in our implementations there is a dispacher. In our implementation there is only one dispacher and one honeypot because the cluster is simple, but the number of honeypots and cluster could vary, depending of the number of sensors and servers.

CHAPTER 5

Implementation Details

5.1 Case of study 1: Cluster and Honeypot for a Ransomware attack

5.1.1 Workflow

After a first phase of research related to honeypots, their taxonomy and features, the work shifted to an actual implementation of at least one of the use cases. In this section the use case regarding a malware honeypot is tackled.
In the very first days, the idea was to consider 4 ESP32 devices as nodes in the cluster. ESP32 is a brilliant device due to its low cost but it lacks some of the features needed for this project, such as a reasonable memory capacity to store enough data files.
Also, due to the high delay in shipping boards, we preferred to use devices that could be granted by Politecnico and LINKS foundation. Therefore, the second option consists in using 4 Raspberry Pi devices.
Once the OS, a lighter distribution of Linux released for Raspberry, got mounted, the actual development started.
At first, a Python script to build a randomly generated file system got released ("createProductionFiles.py"). This script only generates production files and not sentinel files.
Then, a new Python script ("createSentinels.py") was used to generate sentinel files and spread them across the file system.
For each Raspberry, two scripts ("NEWNEWpubsub.py" and "rpiClient.py") provide MQTT and monitoring functionalities.
The last phase of the project focused on the deployment of ransomware to test the defensive mechanism. Literature shows examples of ransomware that can be generated thanks to PowerShell commands, but the final choice was to create an ad-hoc ransomware thanks to a further Python script ("rpiClientPubOnly.py").
The first idea regarding the defensive mechanism was to develop a honeypot on one of the machines in the cluster. This could make all other nodes way simpler and they should only perform functional tasks, exempting them from performing security tasks. On the other hand, if any attack targets the honeypot, then security over the whole cluster is compromised.
The current implementation includes a different solution based on a distributed honeypot in the cluster: each node both performs functional and security tasks. If a node is offline, other nodes will still perform honeypot tasks and security is not compromised.

5.1.2 Structure of the IoT cluster and cluster functionalities

As previously mentioned, the cluster shows a fully connected mesh topology thanks to which each node can communicate to any other in the cluster. This is posssible thanks to MQTT, a communication protocol based on the publish/subscribe paradigm. A node can be a publisher, a subscriber or even both kinds of entities. MQTT is one of the most popular protocols in IoT due to its light header overhead and broadcast features. Subscriber nodes subscribe to a topic and whenever a publisher node communicates a message in the network with the very same topic, such subscribers are able to collect it. A drawback of this approach is the presence of a message broker in the middle of each peer to peer communication, a message broker is an entity, external to the cluster, that receives messages by publishers and forwards them to subscribers. In the case of this project, the message broker is hosted by test.mosquitto.org. Clearly you could think of using a local message broker instead of relying on an external one, improving latency.
Messages exchanged in the cluster can be of two kinds, with the possibility to extend to further kinds:

- Commands. They represent the functional aspect of the cluster taken into account. Normally such messages should be published by the dispatcher in the cluster but it is also possible to assume that in a load balancing cluster a node overloaded with work can delegate another node to perform some of the tasks, thus sending a bash command via MQTT message. This is clearly a vulnerability of the cluster, that the ransomware will try to exploit.
  This kind of message shows the following structure, in JSON:

  ```
  {"src" : <publisher_ID>, "dest" : <dest_node_ID>, "command" : <command>}
  ```

  The topic is

  ```
  "PoliTo/C4ES/<dest_node_ID>/command"
  ```

  It is important to notice that any command dispatched to any of the machine is logged in a text file, ”log.txt” in the toplevel folder of the file system (”fs_creato/”); This feature is compliant to the research purposes of a malware honeypot, that lets the malware move freely in a confined space and records its behavior both for research purposes and for improving security thanks to newly collected information;

- Attack notifications. A node (hereafter referred to as ”victim” node) will broadcast a message of this kind if the periodic check on sentinel files fails. All other nodes in the cluster collect this message and will insert the victim node to their own internal blacklist because the victim’s messages are no longer considered reliable, since an infection is in progress. The victim is deleted from the blacklist aftter a fixed amount of time. This kind of message shows the following structure, in JSON:

  ```
  {"hash_mismatch_in" : <sentinel_file_path>, "untrust_topic" : <victim_node_pubTopic>}
  ```

  The topic is

  ```
  "PoliTo/C4ES/<victim_node_ID>/attack"
  ```

5.1.3 Creation of File System

Production files are files that contain useful data to be elaborated in the cluster. An example could be data gathered from sensors spread in a wide area that are then all forwarded to the nodes in the cluster for some post processing operation. Such post processing operations could be too complex to be performed on the edge by sensor nodes. For instance, you could think of some FIR filters that cut out sensed measures that show too much difference with respect to recent values (it is the case for analog sensed measures, such as temperature).
In this version of the project, production files are filled with random content since nobody will ever operate on data. The file system is filled, in every directory, with a variable number of production files thanks to "createProductionFiles.py". Production files are generated with all random features, both their name and contents, thanks to random() function provided by Python.
Sentinel files, on the other hand, are deployed with the sole purpose of being tripwires for an attacker. Their content, in this version of the project, is also randomly generated. They should always appear on top and bottom of the list of files displayed in a directory. This feature got implemented by ordering production files alphabetically, extracting the first and the last file names and finding names for sentinels (thanks to the ASCII table) such that they would get over the first and last production files. For example, if alphabetically the first and last production files are named "aProductionFile.txt" and "zProductionFile.txt", then two sentinels are named "0aProductionFile.txt" and "zQroductionFile.txt" respectively.
Other sentinels are deployed in the same directory, in the middle of production files, thanks to a logic based on ASCII as well. This system does not aim at filling up the file system with sentinels. On the other hand, its goal is to place them in a smart way. The number of sentinels is proportional to the number of production files, in a 1:3 ratio.
Placing a sentinel at the bottom of the files list is crucial for preventing malicious actions by the malware. If the malware understands that sentinel files are present, a trivial way to avoid encountering sentinel files could be simply reversing the displayed list of files.
Once a sentinel file is created, its hash digest and name are stored in a file, in the very same directory, called ".hashes.txt". This file, once the sentinels are all created, is set in READONLY mode and stores the original hash values for the integrity check on sentinels.
This is a case in which we consider the command "chmod" as our root of trust.

5.1.4 Monitoring Phase

Integrity is the cybersecurity property according to which the content of a file or a message should not change in an undesired manner with respect to the legitimate parties involved in the management of the message or file itself.
Checking the content of all sentinel files in the file system is an operation that is too expensive due to the high time of computation. Moreover, it is a cost that depends linearly on the size of the file to check.
Hash digests provide a simple, yet reliable, way to check the integrity of a file by simply asserting that its hash digest is the same as the one previously stored and kept safe. This is a reasonable statement as long as the original hash digests are kept away from attacks and the hash algorithm cannot be cracked in short time. A way to crack hash algorithms relies on finding a collision. A collision is a couple of plaintext messages that both lead to the same digest. Hash algorithms create a "summary" of the content of a file or message, thus they are always collision prone. SHA512 got chosen as hash algorithm thanks to its strong collision resistance properties.
Integrity check of sentinel files is performed periodically on each raspberry in the cluster. For each directory in the file system, the file ".hashes.txt" is read. For each sentinel present in the file, the

hash digest is recomputed by means of SHA512 and matched against the stored hash. If a mismatch occurs in any of the sentinel files, then the raspberry (i.e., the victim of the attack) will perform two operations:

1. An attack notification is broadcasted to all nodes in the cluster, which will no longer consider the victim as reliable;
2. An immediate shutdown is performed, breaking any possible encryption operation. In the current release, a technician is supposed to restore the status of the victim raspberry, by switching it on again within a fixed amount of time. Once this amount of time has elapsed, the victim node is considered reliable again.

5.1.5 Ransomware

The last phase of the implementation has been reserved to the creation of the actual ransomware for testing the structure.
As previously mentioned, an adoption of a real ransomware got discarded. However, the ad-hoc ransomware shows all the significant operations that a real one would perform.
Also, some enhanced attack features are taken into account. For instance, we supposed that once a ransomware is in the victim's system it might understand that some tripwire files are placed (since it is a well known technique) and therefore might perform some files reordering. Files reordering could arrange files in a way such that production files could be encountered first, compromising the defensive mechanism. Such an arrangement is not possible in our solution since a masking of some bash commands is performed, not allowing harmful commands such as ”sort -R” that would order files in a random way, in a given directory. Command redirection is tackled in the next section of this chapter.
In this cluster structure, the vulnerability that could be exploited by the ransomware is that there is no clear dispatching device to give commands. The ransomware could claim to be a node sending commands legitimately.
The ransomware performs four main operations:

1. A pair of asymmetric RSA keys is generated on its own system;
2. The public key is injected by means of a MQTT message to the victim's system, that has no means to deny the passage of the file;
3. The public key is imported by the victim because the ransomware dispatches a ”gpg –import publicKey” command;
4. The encryption is performed recursively on the whole file system using the public key just injected. This means that the only way to recover files is to collect the private key owned by the ransomware.

5.1.6 Command Redirection

Since the response to malware attack relies on sentinel files we have to increase the probability a sentinel file is encrypted as first in the list. Unfortunately, as previously mentioned, there are linux commands that could be used to change the order of files; for example, the simple command ”ls — sort -R” will randomly sort the output of the list command making the naming strategy for files in some case useless as a first strategy protection. In this case, masking the ”sort -R” command we can provide a resulting order that is suitable for this kind of approach as well. In our case the masking procedure relies in some python scripts, one for each command, used than when defining alias of

commands directly in /etc/bash.bashrc file that is executed each time a new bash is open.
Masks for command implemented are:

- sort command: the output from 'sort' command will be changed in case of a use of the option -R, as previusly stated, in this case sentinel files are inserted as first and last elements of the list, while remaining sentinels are equally distributed.
- list command: a problematic usage of the 'ls' command is with option -t, this is due to sentinels files that are not often updated, potentially never, and in this may lead for them to appear always at the end of the list; also in this case the list is modified to have the first and the last element as a sentinel and the others equally distributed. Another strategy could be updating timing informations on sentinel files with 'touch' command before calling the list command, this is left as future work.
- remove command: 'rm' command is problematic since the option -f allows to remove files with a read-only protection, in this case the usage of rm -f is forbidden, if selected it will perform nothing.

Of course, many other modifications are possibile with linux commands, a more precise analysis could be helpful in protecting the system. In this implementation the command masking strategy is intended to be tightly related with the shallow ransomware that will shows it's function. Commands are not intended to work in every condition nor it is guaranted someone will success sorting files relying on single files properties. For example, listing all files with absolute path and then sorting them randomly or even worse based on the last modification will not be taken into account as a strategy protection, for these cases the research honeypot strategy of storing all commands in the log file to further analysis still remains valid showing the real final objective of this type of honeypot implementation.

5.2 Case of study 2: Cluster and Honeypot for a DoS attack

In this part of the project we first implemented a cluster IoT which communicates with clients via sockets then adding the cyber security part: the dispatcher and the honeypot.

5.2.1 Instruments used

The most important aspect to respect when developing a client-server application, like our ioT cluster, it that the servers and the sensors have to be able to exchange data between each others. The module sockets present in the python library allow us to support networking protocols between two or more processes across machines. In this particular case we use TCP sockets, where at one side a process act like a client and on the other side a process act like a server. Also python offer the module selectors that allow us to serve multiple sockets connections. In particular the .select() method allows us to check for I/O completion on more than one socket. So we can call .select() to see which sockets have I/O ready for reading and/or writing. We also want to underline that with .select() we're not able to run concurrently.

5.2.2 First structure of the IoT cluster

In our first simulation we decided to implement an high-interaction honeypot and a cluster reported in the figure below.

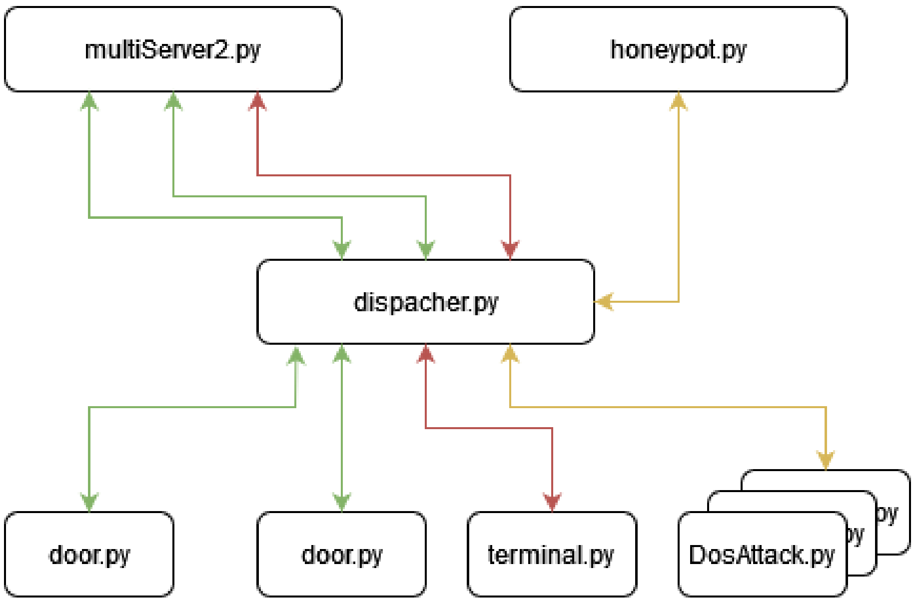

Figure 5.1: Our simple IoT cluster with an high-interaction honeypot

We have different type of client (just like in the reality of an heterogeneous IoT cluster). In the nexts subsections we will explain the purpose of every script and how they interact. We would like to remind that on the git repository there is also a tutorial about the use of the first version of the cluster. For this implementation we started with an example of scripts for client and server that as the first prototypes of our cluster, the clients were then modified to diversify and create the possible sensors of our cluster (in this case a door and a terminal), while the server has been modified to handle more than one connection.

Door.py

This script represent a door that could be controlled by our system. It have a global variable named status , that could assume 2 values, UNLOCK and LOCK. This are the status of the door. When the "sensor" is turned on, it send 2 packets to the server with the initial status of the door, then it wait for commands from the server. The packets send by the door sensor have this format:

```
{topic: DOOR data: LOCK/UNLOCK IPFALSE: 129.0.0.1}
```

to mask the loopbacks IP with external IP, but then we understand that this was not necessary. A socket can be distinguished by another by the couple (IP,port) of the client and the server. When the scripts from this implementation will be launch from different network, the system will behave in the same way of our tests on loopback. We decided to leave this field because if in the future we need to exchange more data, we could recycle it, of course changhing the name. The data that the door could receive from the server have this format:

```
{topic: SERVER data: STATUS/LOCK/UNLOCK IPFALSE: someData}
```

request the status, the door check the variable STATUS and sent it value to the server. If the command is LOCK the STATUS variable is set to LOCK, if the command is UNLOCK the STATUS variable is set to UNLOCK.

Terminal.py

This script represent a terminal where the admin of the IoT cluster could control the sensors and check the status of the system. The terminal offer a set of command to visualize the number of client connected to the cluster, send command to control them and receave all the status of all the clients, a better description is reported in the user manual.
The terminal at the begin for the system is a normal client so is always check by the Honeypot if it is trusted after that it can starts to send packets from the user. The packets sended from the terminal has the same structure described before, the user can decide the status (in this case for the door LOCK/UNLOCK). The structure of the packets from the server to the terminal depends from the command sended by the terminal. Here we report some example:

1. if the terminal want to open the door 1, the server will return :

   ```
   {topic: SERVER data: OPEND data2: UNLOCK  IPFALSE: someData}
   ```

2. if the terminal want to close the door 2, the server will return :

   ```
   {topic: SERVER data: CLOSED data2: LOCK  IPFALSE: someData}
   ```

3. if the terminal want to see the status of door 2, the server will return :

   ```
   {topic: SERVER data: GETSTATUSD data2: 2\_LOCK/UNLKOCK  IPFALSE: someData}
   ```

As it is implemented, if the terminal do not receive a response from the server, it stops working. In future version we could add a timer, and ,after some time elapsed, we stop wainting from the response and rise an error.

MultiServer2.py

The server have to handle all the connection of the sensors. It have to connect the terminal to the doors, and send the command to the correct door. If the command is for all the doors connected, it creates and sends packets with the command to all of them. It has a list of sockets of the doors and a list for the socket of the terminals, when it needs to send a packet checks always that list to find the istance of the right socket, each socket comunicates with the dispatcher this because the server needs to be reached only by the packtes from the dispacher, there is no connection between the server and the client, so each time a new connection arrives a new socket is created between the server and the dispatcher for that specific client. As it is implemented, our system work with only one terminal connected to the server, but this could be ealisy changed.

Dispacher.py

Created after multiserver and cluster in order to have in mind how packets should communicate and their structure, the dispacher is the first line of defense with our honeypot. All the new connection from outside the network are sent to the dispacher. Here we have 3 list in python, one for the trusted socket named socketWhiteList, that could communicate with the server. Every client in the white list have a personal socket from the dispacher to the server, the couple is saved in the list coppiaSocketWhiteListSocketServer. The list socketPending save all the new connection from outside that are waiting from the response of the honeypot. When a new connection arrive to the dispacher, the packet sent are redirect to the honeypot. The packet from the dispacher to the honeypot are sent in the same socket named socket_honey. When a response from the honeypot arrive, if the connection is approved the dispacher move the socket from the list of the pending one to the list of the whitesocket.

If the connection is not approved the sockert is delete from the list of the pending one and closed forever. When a new connection arrive, the dispache sent to the honeypot first the peer name of the new connection, and then the first packet received from it. While a connection is pending, all the other packets from outside are lost, it's just an implementation choise.

DosAttack.py

The script for the Dos attack is very simple, it is called in a sequential way from DosAttack.sh.
After establish a connection with the dispatcher, through an infinite loop, the script starts sending each second random data (in a wrong structure) using the socket created for data exchange.

Honeypot.py

The honeypot is the last element implemented for the system. It controls the packets of the new connection sended by the dispacher. If they came from a terminal or a door (and so with a trusted structure) , the new connection is approved , otherwise is not approved. The format of the data from the honeypot to the dispacher is:

`{"ip-port: (ip, port) status: TRUSTED/UNTRUSTED"}`

The honeypot in principle handles only one request of checking the new connection, after a first test of the system we have decided that it could work with multiple connections.

5.2.3 Second structure of the IoT cluster

After the presentation of our projects it was notice to our group that the honeypot and the dispacher could be collapsed in just one "server" , actualy a script, that handle all the function of the dispacher and the honeypot. So we create the script dispacherAndPot.py . Here, we have a function named honeypot, that , like the reader probably already ìunderstand, is the honeypot. When a new connection arrive, this function is called and check the packets sent to the dispacher. If it is ok it return 1, otherwise 0. This low-interaction is much more efficient and simple, it do not lose packet from pending connection and we do not neeed to create socket between the honeypot and the dispacher. According to the complexity of the simulation right now, this implementation is much better, but if we want for example to improve the honeypot with new feature, like the ones from our other implementation, we suggest to use the first version. The picture below report the structure of the second implementation.

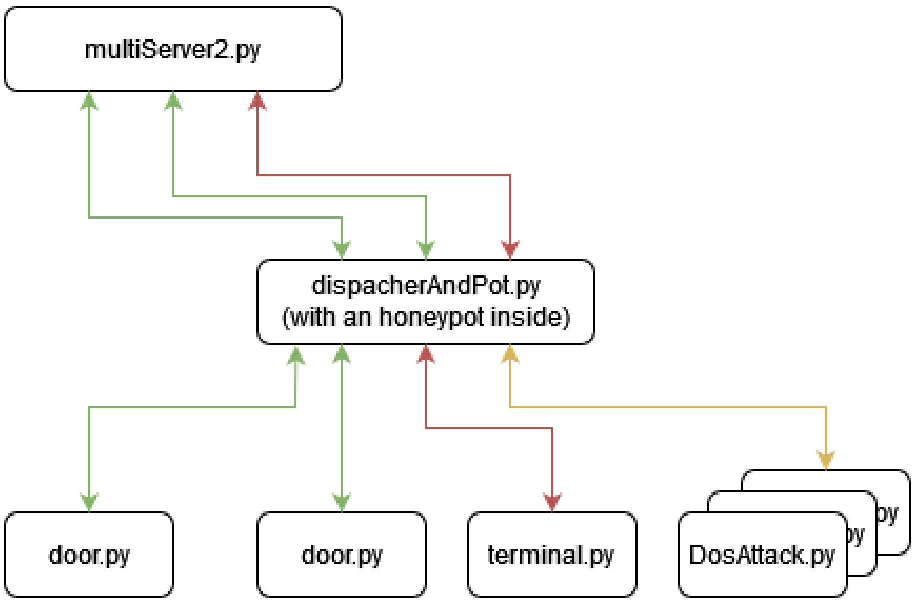

Figure 5.2: Our simple IoT cluster with a low-interaction honeypot

CHAPTER 6

Results

6.0.1 Case of study 1: Cluster and Honeypot for a Ransomware attack

Results show that the solution is effective since most of the file system is spared from the encryption. Malware honeypots belong to the wider category of research honeypots. The behavior of the malware should be tracked in order to gather information and eventually create better ad-hoc countermeasures. Moreover, the system of insertion of infected nodes into a blacklist stops the spread of the malware, which is one of the crucial points to tackle when dealing with malware and their power of infection.

6.0.2 Case of study 2: Cluster and Honeypot for a DoS attack

High level version

We run the DosAttack.sh with different number of scripts launched, 50,100,150,200 and 400. For this version, the cluster with the honeypot was able to handle up to 200 fake sensors. Unfortunately with 400 Dos scripts the virtual machine where the system is simulated wasn't able to establish all the sockets, due to the high number of resources requested.

Low level version

We run the DosAttack.sh with different number of scripts launched, 50,100,150,200 and 400. For this version, the cluster with the honeypot inside the dispatcher was able to handle up to 400 fake sensors. This version works better than the other because the dispatcher don't have to redirect the data from the Dos sensors to the honeypot, because it is instantiated inside it.
Results obtained are two solutions for two different attacks on two complementary clusters, a P2P cluster where therefore all the nodes are the same and talk to each other without hierarchies and a cluster that works in a very hierarchical way, with a main server and all the others that they accept connections from clients through a dispatcher. Same for the two attacks that are also completely different from each other as we have fully explained in the previous chapters.
Testing the DoS solution we noticed that it is able to manage small DoS attacks (this is also due to the virtual machine which obviously does not allow an exhaustive test). If attacked, our solution manages to perfectly manage 100 attackers, without creating delay to the server, reaching up to ...

6.1 Known Issues

6.1.1 Case of study 1: Cluster and Honeypot for a Ransomware attack

- The masking of commands is ready and functional, but the commands dispatched by the ransomware cannot fully exploit it. So far the only strategy adopted is considering an encryption procedure capable of encrypting files retrieved in a recursive manner with the 'find' command, thus in alphabetical order. Files we are delivering will include the code that was intended to show the behavior of commands masking procedure. Unfortunately, sending this long command through MQTT will broke it's the execution for some reason that we were not able to solve in a complete way. Specifically the malwere command will recursively list all files applying then the very same encryption mechanism as before. The custom visit to the system directory allows the insertion of the specific masked commands. Although this is an issue it is now the main target for future works at the moment of deliverance;
- JSON decode function returns a decode error, once in a while, when reading the file "blacklist.json", this only happens rarely.

6.1.2 Case of study 2: Cluster and Honeypot for a DoS attack

There are many way to improve our simulation before deploy it on a real hardware implementation. First we need to model more sensor. Then we need to encript and decript the data in the socket, otherwise the attacher could understand the ID of the sensors and invalidate the honeypot. Then we have to create a server that works on multiple thread and that is able to speak with other servers.

Hardware Limitation

The Dos solution has suffered a lot from the limited number of raspberry available this because, differently from the solution developed in parallel, it needs two additional raspberries for dispatchers and honeypots that behave differently from all of them, for this reason the solution is limited to a single server , dispatcher, honeypot, and two types of clients.

6.2 Future Work

6.2.1 Case of study 1: Cluster and Honeypot for a Ransomware attack

In a later release of this project we could develop a more realistic ransomware, also considering issues of the actual implementation, or even deploy a real one and release it on a virtualized space to limit damages.
We could also shift from a polling monitoring to a smarter one that needs less effort. For instance, we could just monitor the integrity of sentinel files after a command is dispatched. There is also the possibility of wrapping linux system calls directly gaining a more precise way to manage and check sentinel files, in this case the root of trust will be the kernel space of the operating system.

6.2.2 Case of study 2: Cluster and Honeypot for a DoS attack

There are many way to improve our simulation before deploy it on a real hardware implementation. First we need to model more sensor. Then we need to encript and decript the data in the socket, otherwise the attacher could understand the ID of the sensors and invalidate the honeypot. Then we have to create a server that works on multiple thread and that is able to speak with other servers.

CHAPTER 7

Conclusions

In this project we made a survey of honeypot categories and implementations as for the current state of the art. We developed then two solutions targeting malware and DoS attacks respectively. For the first one we implemented an IoT cluster based on rasberry pi systems exploting MQTT communication protocol to send and receive bash commands. We showed how malwares, ransomware specifically, could interact in such a system and proposed some common countermeasures to the problem. Results of the honeypot research have been stored in specific log files that could be then used to update the system. We believe this approach to be good in discovering, then targeting, new ransomware attacks thus improving existing countermearuses. For the second proposal we implemented a tree cluster able to protect the system from DoS attack thanks to a dispatcher which protects the server first creating different socket for different client and managing their packets and then by filtering unwanted packets by sending them to the honeypot which checks that the ip is not blacklisted and that it is sending sane packets, if not it will blacklist it and mark its connection as untrasted.

APPENDIX A

User Manual for Malware Honeypot

In this User manual it is given a brief explanation of how to raise up the cluster and launch the attack. For this Project python is requested, if it is already installed skipp the following section.

A.1 How to Install python

For this project is requested a version of python up to 3.X.X. To install python and the used package in the environment of each Raspberry Pi follow this steps:

1. Install the required packages for python with this command

```
\$ sudo apt install build-essential zlib1g-dev libncurses5-
dev libgdbm-dev
 libnss3-dev libssl-dev libreadline-dev libffi-dev wget
```

2. Install python (in this case 3.8)

```
\$ sudo apt install python3.8
```

   To check if the correct version of python is installed the following command can be inserted:

```
\$ python --version
```

3. Install pip3 package with the command

```
\$ sudo apt install python3-pip
```

4. Install the PAHO MQTT library with the command

```
\$ pip3 install paho-mqtt
```

A.2 How to set the cluster environment

After connecting throw ssh with the raspberry Pi and cloning the repository on each Raspberry Pi follow this passage for each of them:

1. Enter in the folder ”fs_creato” with the command

```
\$ cd DELIVERY/NODE_DELIVERY/fs_creato
```

2. Type the following two commands, the first is needed to specify the interpreter for the bash script while the other creates the file system

```
~/DELIVERY/NODE_DELIVERY/fs_creato $ sed -i -e 's/\r$//' script2.sh

~/DELIVERY/NODE_DELIVERY/fs_creato $ chmod +777 ./script2.sh
```

At the end It has to be displayed in the terminal of the Raspberry Pi

```
pi<number_of_the_pi> created
subscribed to PoliTo/C4ES/#
```

A.3 How to lauch the malware attack

After cloning the repository in the device is used for this purpose follow this steps:

1. Enter in the folder "MALWARE_DELIVERY" with the command

```
\$ cd DELIVERY/MALWARE_DELIVERY
```

2. Type the following two commands again the first is needed to specify the interpreter for the bash script while the other creates the malware

```
~/DELIVERY/MALWARE_DELIVERY $ sed -i -e 's/\r$//' script.sh

~/DELIVERY/MALWARE_DELIVERY $ chmod +777 ./script.sh
```

At the end It has to be displayed in the terminal of the Raspberry Pi

```
MALWARE created
subscribed to PoliTo/C4ES/#
```

And it starts to inject the public key into the File system after a while a message

```
operation completed successfully
```

When the encryption starts from the malware a message is shown in the terminal of the Raspberry Pi attacked regarding a mismatch of the sentinel file, the procedure is shut down and the connection closed.

```
Start shutdown procedure RPI...
blacklist cleared
Connection to <IP> closed by remote host.
Connection to <IP> closed.
```

It can be also checked the insertion inside the blacklist in all the other Raspberry Pi just checking the terminal connected (via ssh) to the others Raspberry Pi.

A.4 Plus: How to check how many files have been encrypted

It is a feature provided to check the effective number of encrypted file.

1. Enter in the folder "NODE_DELIVERY" with the command

   ```
   \$ cd DELIVERY/NODE_DELIVERY
   ```

2. Insert the command

   ```
   \$  ~/DELIVERY/NODE_DELIVERY $ python3 ./count.py
   ```

   to check how many files have been encrypted and which one.

A.5 Plus: How to visualize the log file

1. Enter in the folder "fs_creato" with the command

   ```
   \$ cd DELIVERY/NODE_DELIVERY/fs_creato
   ```

2. Insert the command

   ```
   \$  ~/DELIVERY/NODE_DELIVERY/fs_creato $ less log.txt
   ```

   It is possible to see the name of the node that perform the encryption, the public key and the steps it performs.

APPENDIX B

User Manual for DOS Honeypot

In this user manual we will see how to setup the cluster, some of its features to control and check its status and how to inject a DOS attack inside of it.

B.1 How to start the cluster., version High Level

After downloading and unzipping the folder from GIT in order to run the simulation in the correct way we need to open at least four shell and follow this step:

1. In the first shell launch the honeypot with the command below, the 2 argument are the Ip and the port number. They are just an example , but it must be the same of the values saved in dispacher.py in the global variable HOST_H and PORT_H so if is necessary to change it, they must be changed inside the code.

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python honeypot.py 127.0.0.2 65430
   ```

2. In the second start the server with the command below with the argument equal to the value HOST_S and PORT_S in the dispacher.py even here if we want to change it is necessary to change it in the code.

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python multiServer2.py 127.0.0.3 65429
   ```

3. In the third launch the dispacher with the command

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python dispacher.py 127.0.0.1 65430
   ```

4. In the others shell finally start all the sensors, in order to connect them to the dispacher, the global variable in all the sensors and Dos Attack HOST and PORT must be the same of the argument given to the dispacher

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python <nameofthesensor>.py
   ```

   It is possible to check if the honeypot and dispatcher works correctly, it must be in the shell honeypot and dispatcher the ip and port of the client and the status (in this case TRUSTED). Another possible check that is possible to do is to run (as the client) terminal.py and with the command

   ```
   \$ LISTD
   ```

B.2 How to start the cluster., version Low Level

After downloading and unzipping the folder from GIT in order to run the simulation in the correct way we need to open at least four shell and follow this step:

1. First we start the server with the command below with the argument equal to the value HOST_S and PORT_S in the dispacherAndPot.py even here if we want to change it is necessary to change it in the code.

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python multiServer2.py 127.0.0.3 65429
   ```

2. In the third launch the dispacher with the honeypot with the command

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python dispacherAndPot.py 127.0.0.1 65430
   ```

3. In the others shell finally start all the sensors, in order to connect them to the dispacher, the global variable in all the sensors and Dos Attack HOST and PORT must be the same of the argument given to the dispacherAndPot

   ```
   ~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ python <nameofthesensor>.py
   ```

   It is possible to check if the honeypot and dispatcher works correctly, it must be in the shell honeypot and dispatcher the ip and port of the client and the status (in this case TRUSTED). Another possible check that is possible to do is to run (as the client) terminal.py and with the command

   ```
   \$ LISTD
   ```

B.3 How to use the terminal

Here the list of the possible command for the terminal and what they do:

- **LISTD**: it return the number of door connected to the system, it do not require other argument. In our system to refer a specific door we use a number to identify it. So if 2 doors are connected to the server, to act on the first door we digit 1, to act on the second we digit 2, to act on all trhe doors we digit ALL. The format for the packes from the terminal to the server for this command is :”topic: TERMINAL data: LISTD data2: bho IPFALSE: 130.0.0.1”. The field data2 is used by the other commands

- **OPEND**: it allow the user to open a specific door connected to the system or open all the doors. For example, if we want to open only the door 2 we digit OPEND 2, if we want to open all the doors we digit OPEND ALL . The format for the packes from the terminal to the server for this command is :”topic: TERMINAL data: OPEND data2: 1/2/ecc../ALL IPFALSE: 130.0.0.1”.

- **CLOSED**: it allow the user to close a specific door connected to the sytem or close all the doors. For example, if we want to close only the door 2 we digit CLOSED 2, if we want to close all the doors we digit CLOSED ALL . The format for the packes from the terminal to the server for this command is :”topic: TERMINAL data: CLOSED data2: 1/2/ecc../ALL IPFALSE: 130.0.0.1”.

- **GETSTATUSD**: it allow the user to know the status of all the doors connected to the system. The format for the packes from the terminal to the server for this command is :”topic: TERMINAL data: GETSTATUSD data2: 1/2/ecc../ALL IPFALSE: 130.0.0.1”. We could check the status of a specific door or of all the doors.

After a coomand is sent to the server, our terminal start waiting for the response. The returned data depends from the command wrote to the server.

- **data for LISTD**: the data format returned by the server is : "topic: SERVER data: LISTD data2: 'NumberOfPortConnected' IPFALSE: 128.0.0.1" . Thew termianl print the number of door available.
- **OPEND**: the data format returned by the server is : "topic: SERVER data: OPEND data2: 'statusOfTheDoor' IPFALSE: 128.0.0.1" . The terminal then show the status of the door selected or off all the door selected
- **CLOSED**: the data format returned by the server is : "topic: SERVER data: CLOSED data2: 'statusOfTheDoor' IPFALSE: 128.0.0.1" . The terminal then show the status of the door selected or off all the door selected
- **GETSTATUSD**: the data format returned by the server is : "topic: SERVER data: GETSTATUSD data2: IDDoor_statusDoor IPFALSE: 128.0.0.1" . The terminal then show the status of the door selected or of all the door selected .

B.4 Inject the DOS attack

After the first phase of build of the cluster with all the client it is possible to attack the system with the script DosAttack.sh

```
~/pythonImplementation\_29\_06\_HIGHLEVELHONEY\$ ./DosAttack.sh
```

It is possible to check what the script doing just looking on the dispatcher terminal looking how the cluster manage the attack. To checge the value of Dos shell that attack the system just modify DosAttack.sh .

Bibliography

[1] 2018-2022 ransomware statistics and facts. https://www.comparitech.com/antivirus/ransomware-statistics/.

[2] 73 ransomware statistics vital for security in 2022. https://www.pandasecurity.com/en/mediacenter/security/ransomware-statistics/#:~:text=The%202%2C084%20ransomware%20complaints%20received,(FBI%20and%20CISA).

[3] Ransomware reality shock: 92% who pay don't get their data back. https://www.forbes.com/sites/daveywinder/2021/05/02/ransomware-reality-shock-92-who-pay-dont-get-their-data-back/?sh=20d9faf3e0c7/.

[4] Hyrum S. Anderson and Phil Roth. Ember: An open dataset for training static pe malware machine learning models, 2018.

[5] M Anirudh, S Arul Thileeban, and Daniel Jeswin Nallathambi. Use of honeypots for mitigating dos attacks targeted on iot networks. In *2017 International Conference on Computer, Communication and Signal Processing (ICCCSP)*, pages 1–4, 2017.

[6] Anujot Boparai, Ron Ruhl, Dale Lindskog. The behavioural study of low interaction honeypots: Dshield and glastopf in various web attacks, 2014. [Concordia University College of Alberta 7128 Ada Boulevard, Edmonton, AB T5B 4E4, Canada].

[7] He Chen, Qian Wang, Parthajit Mohapatra, and Nikolaos Pappas. Secure status updates under eavesdropping: Age of information-based physical layer security metrics, 2020.

[8] Rajat Gupta, Madhu Viswanatham V., and Manikandan K. An innovative security strategy using reactive web application honeypot, 2021.

[9] Francesco Malandrino, Alessandro Nordio, and Carla Fabiana Chiasserini. Eavesdropping with intelligent reflective surfaces: Threats and defense strategies.

[10] Niels Provos. Honeyd : A virtual honeypot daemon (extended abstract). 2003.

[11] Yulong Zou, Jia Zhu, Xuelong Li, and Lajos Hanzo. Relay selection for wireless communications against eavesdropping: a security-reliability trade-off perspective. *IEEE Network*, 30(5):74–79, sep 2016.