



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona



Design and implementation of a semi-automated threat analysis system -Tartarus-

A Master's Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Rubén Barceló Armada

In partial fulfilment
of the requirements for the
Master's Degree in Cybersecurity

Advisor: Pere Barlet Ros
Barcelona, June 2022

Abstract

In recent years, there has been a notable increase in the number of cyberattacks caused by the change in the work model due to the pandemic. People were forced to work at home, away from the secure confines of a corporate network. Moreover, in the coming years the number, intensity and variety of these attacks are expected to keep increasing. Therefore, it is of vital importance to constantly track and adapt to the new techniques, vulnerabilities and malware used by the cybercriminals to understand your level of exposure against these new threats.

This project aims to implement a semi-automated threat analysis system that can distribute and execute malware samples into sandboxed virtual machines. Once it has been executed, this system retrieves metrics of the results of each execution to observe the detection and blocking capacity of the machine against these threats.

Acknowledgments

I would like to thank all the people who, directly or indirectly, have been involved in this project.

First of all, my project manager (*Carlos Fernández Sánchez*) for guiding and helping me in the development of this project.

Secondly, I would also like to thank the entire Incide team, especially *Iker Loidi* and *Carlos Navarro*, for always be willing to help me.

Third, my colleague from the masters, *Pablo Aznar*, for giving his opinion about some doubts I had during the development of this project.

Lastly, I would like to thank my family and partner for supporting and motivating me in the most difficult moments during all the master's degree.

Table of contents

Abstract	2
Acknowledgments.....	3
1 Introduction and contextualization	9
1.1 Introduction.....	9
1.2 Contextualization	9
2 Project scope	10
2.1 Main objective.....	10
2.2 Sub-objectives	10
3 State of the art of the technology used in this thesis.....	11
3.1 State of the art	11
3.2 Technology used.....	11
4 Methodology	16
4.1 Development method	16
4.2 Validation method.....	16
5 Work Plan.....	17
5.1 Tasks	17
5.2 Initial Planning.....	18
5.3 Final Planning	20
6 Budget.....	22
6.1 Human costs	22
6.2 Generic costs	23
6.3 Contingency.....	23
6.4 Total cost	24
7 Project development	25
7.1 Modules.....	25
7.2 Infrastructure of the application	26
7.3 Network.....	27
8 Module #1 - Infrastructure	28
8.1 Introduction.....	28
8.2 Components	28
8.3 Functionalities	28
8.4 Database tables.....	29
8.5 Previous configuration	29
8.6 Images	31

8.7	Virtual Machines.....	33
8.8	Snapshots	37
8.9	Network.....	39
9	Module #2 - Simulation	45
9.1	Introduction.....	45
9.2	Components	45
9.3	Database tables	45
9.4	Tasks	46
9.5	Agent	49
10	Module #3 – Reporting	52
10.1	Introduction.....	52
10.2	Fleet.....	52
10.3	Kibana	53
11	Analysis of the results	56
11.1	LockBit 2.0	56
11.2	Follina	60
12	Conclusion.....	66
12.1	Future development.....	66
13	Bibliography.....	68

List of Figures

Figure 1 - I4Tables overview	12
Figure 2 - I4Tables configuration file.....	12
Figure 3 - Command to launch I4Tables tool	12
Figure 4 - Example of allowed and denied domains	13
Figure 5 – Example of Vagrantfile	13
Figure 6 - Tartarus CLI	14
Figure 7 - Initial Gantt diagram	19
Figure 8 - Final Gantt diagram.....	21
Figure 9 - Tartarus architecture	26
Figure 10 – Overview of Tartarus network	27
Figure 11 - Infrastructure use case diagram	29
Figure 12 - Infrastructure DB Tables	29
Figure 13 - Disk2vhd interface	30
Figure 14 - Images use case diagram	31
Figure 15 - Add image	31
Figure 16 - Modify image	32
Figure 17 - Delete image	32
Figure 18 - List images.....	33
Figure 19 - VM use case diagram	33
Figure 20 - Create virtual machine	34
Figure 21 - Delete virtual machine	34
Figure 22 - List virtual machines.....	35
Figure 23 - Start virtual machine.....	35
Figure 24 - Stop virtual machine	36
Figure 25 - Restart virtual machine	36
Figure 26 - Snapshots use case diagram	37
Figure 27 - Take snapshot	37
Figure 28 - Revert snapshot	38
Figure 29 - List snapshots	38
Figure 30 - Delete snapshot	39
Figure 31 - Network use case diagram.....	39
Figure 32 - Add network image	40
Figure 33 - Delete network image.....	40
Figure 34 - Start network virtual machine	41
Figure 35 - Stop network virtual machine	41
Figure 36 - Restart network virtual machine	41
Figure 37 - Create profile	42
Figure 38 - Delete profile	42
Figure 39 - List profiles	43
Figure 40 - Start I4Tables tool	43
Figure 41- Stop I4Tables tool.....	43
Figure 42 - Status I4Tables tool.....	44
Figure 43 - Simulation DB Tables.....	45
Figure 44 - Task specification	46
Figure 45 - Tasks use case diagram	46

Figure 46 - Add task.....	47
Figure 47 - Import task.....	47
Figure 48 - Cancel task.....	48
Figure 49 - Delete task.....	48
Figure 50 - List tasks.....	49
Figure 51 - Operation of Tartarus agent.....	49
Figure 52 - Example of the operation of an agent.....	50
Figure 53 - Agents use case diagram.....	50
Figure 54 - Elastic agent policies.....	52
Figure 55 - Windows agent integrations.....	53
Figure 56 - Elastic agents.....	53
Figure 57 - Event display page.....	54
Figure 58 - Alert display page.....	54
Figure 59 - Alert process tree.....	55
Figure 60 - rules file for Lockbit simulation.....	56
Figure 61 - iplist file for Lockbit simulation.....	56
Figure 62 – Creation of LockBit profile.....	57
Figure 63 - Network VM startup.....	57
Figure 64 - I4Tables tool startup.....	57
Figure 65 - VM startup.....	57
Figure 66 - Overview of the VMs.....	57
Figure 67 - LockBit task specifications.....	58
Figure 68 - LockBit task imported into Tartarus.....	58
Figure 69 – Wallpaper modified by LockBit.....	58
Figure 70 – Alerts generated by LockBit.....	59
Figure 71 - LockBit process flow.....	59
Figure 72 – General LockBit events.....	60
Figure 73 - LockBit file events.....	60
Figure 74 - iplist file for Follina simulation.....	61
Figure 75 - rules file for Follina simulation.....	61
Figure 76 - Overview of the VMs.....	61
Figure 77 - Follina task specification.....	62
Figure 78 - Follina task imported into Tartarus.....	62
Figure 79 - Word document retrieving an HTML file from a webserver.....	62
Figure 80 - Microsoft Windows Support Diagnostic Tool opened by a Word document.....	63
Figure 81 - File created by Follina exploit.....	63
Figure 82 – Alert generated by Follina.....	64
Figure 83 - Follina process flow.....	64
Figure 84 - Details of cmd.exe process.....	64
Figure 85 - File events of cmd.exe process.....	65
Figure 86 - Network log generated during Follina execution.....	65
Figure 87 - Details of network log.....	65

List of Tables

Table 1 - Initial estimated time	19
Table 2 - Final planning	20
Table 3 - Human costs	22
Table 4 - Cost per activity	23
Table 5 - Generic cost	23
Table 6 - Total cost	24

1 | Introduction and contextualization

1.1 Introduction

With the arrival of COVID-19 pandemic, almost all countries had to apply restrictions and lockdowns to their population. This caused the work model to change radically, where a large majority of people had to start working from their homes, with the cybersecurity implications that this fact entails for both the employees and the organizations.

Cyber-attackers see the pandemic as an opportunity to step up their criminal activities by exploiting the vulnerability of employees that were working at home. As detailed in the report *Cyber Attack Trends by Check Point*[1], global cyberattacks increased by 29% and Ransomware attacks surged 93% in the last year.

Therefore, it is of vital importance that companies constantly track, monitor, learn and adapt to new vulnerabilities and malware that is being used by the Advanced Persistent Threat groups (APTs) or other cybercriminals. This process allows them to understand their level of exposure and blocking capacity against new threats and prioritize the mitigations that need to be implemented.

1.2 Contextualization

This master thesis has been developed jointly with Incide[2], where I have been doing an internship while developing this project. Incide is a company specialized in the field of cybersecurity. It has a reactive services' area which includes Incident Response and Digital Forensics. At the same time, they also have an area of preventive services based on audits, Red Team exercises, improvement of detection systems, among others.

One of the services offered by this company that has been rising in recent years is the *Imminent Threat Exposure (ITE)*, which provides to the customer a quick overview of their threat exposure. This service is based on constantly tracking new techniques and vulnerabilities used by cybercriminals. Once a new threat has been detected, Incide obtains or creates a sample of the threat to perform a simulation in a controlled environment of the client's infrastructure. Finally, Incide checks the detection and blocking capacity of the systems against this malware or vulnerability.

However, currently all the processes related to this service are performed manually by an analyst, which does not allow proper scaling, parallelization and automation of the tasks.

2 | Project scope

Once the project is contextualized, the objectives that will allow us to solve the problem presented above will be defined.

2.1 Main objective

The main goal of this project is to design and implement a system, called Tartarus, which can distribute and execute malware samples into sandboxed virtual machines.

The system will also retrieve metrics of the results of each execution, as well as indicators of compromise and behaviour to observe the detection and blocking capacity of the system, the installed EDR and the implemented security policies.

2.2 Sub-objectives

The following sub-objectives must be resolved to achieve the objective stated above:

- 1. Manage infrastructure.** Design and develop the system to manage the infrastructure where the threat simulations will be performed. This includes the virtual disk images of the clients' workstations, the sandboxed virtual machines and snapshots.
- 2. Manage the deployment of the malware samples.** Design and develop the system to reproduce the imminent threats in the virtualized workstation.
- 3. Retrieve metrics of the result.** Design and develop the system to observe the detection and blocking capacity of the virtualized workstation against a threat.

3 | State of the art of the technology used in this thesis

3.1 State of the art

A few years ago, Incide implemented an application called Nemesis that had a similar objective to the one developed in this project. However, it had certain negative aspects that did not allow them to continue its use.

Nemesis was developed for a specific client and consequently, this application had to fulfil the requirements that this client needed for his own company. In addition, due to the state above, the virtual machines that this tool could host were limited. Finally, Nemesis was too complex and focused on a single use case, making it very difficult to maintain, manage and add new functionalities.

Therefore, Incide wanted to re-implement the tool from scratch following the idea of Nemesis but, in this case, designing and developing an application that is focused on automating the Imminent Threat Exposure service mentioned above.

3.2 Technology used

This section presents the tools that have been used for the development of Tartarus.

3.2.1 *Isolated Local Network*

One of the most important steps when you are trying to simulate a real attack scenario is to isolate the machine where you are going to execute the malware, so that any other computer or system in your network can be infected or in danger. However, we also need to consider that there are some connections that are required to execute the malware correctly.

To solve this problem, we will use a tool called **I4Tables** that was developed by Iker Loidi from Incide. This is a python tool that acts both as a firewall and DNS; allowing to filter all the communications and only letting through those packets that are permitted in a rule's configuration file. Indeed, I4Tables will translate the rules into Linux iptables rules that will be added on the machine where it is running.

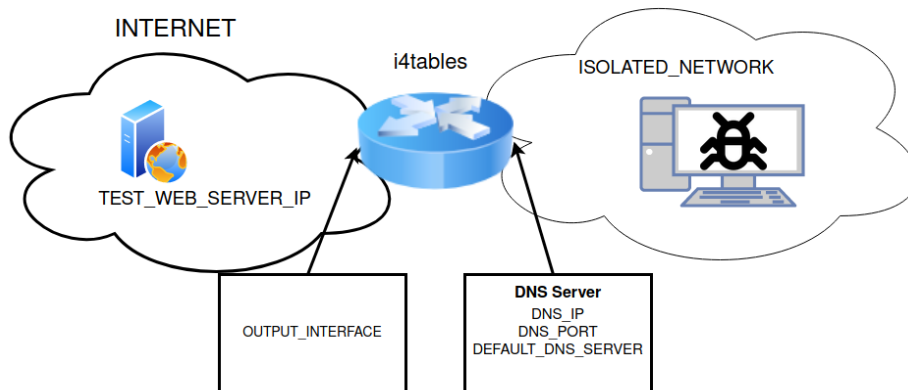


Figure 1 - I4Tables overview

As mentioned above, to use this tool the user must provide a rule's configuration file and an iplist file. On one hand, the rule's file includes the domains and subdomains that the user wants to permit communication with. We can specify dynamic (*a0.awsstatic.com*) or static (*incide-server.local; 192.168.101.1*) domain resolutions. On the other hand, an iplist file which includes a list of IPs that the virtual machine connected to I4Tables must perform a *ping* to confirm that it is isolated.

```

Abrir  [+]  rules.txt
~/Documentos/TFM/vm/i4tables/i4tables/profiles/conti-xml
1 # File to configure i4tables resolutions
2 incide-server.local;192.168.101.1
3 movekochd.com;192.168.103.10
4 a0.awsstatic.com

```

Figure 2 - I4Tables configuration file

In the following image you can observe the command to launch I4Tables application. From this moment on, the tool will wait for a computer to start the validation process to connect to it.

```

i4tables@i4tables:~/vagrant/i4tables$ sudo python3 i4tables.py --rules profiles/default/rules.txt
plist profiles/default/iplist.txt
*****
***** REGLAS APLICADAS *****
*****
[STATIC_RULE] incide-server.local -> 192.168.101.1

[+] DNS Listening on 192.168.103.10:53 ...
[challenge] Validacion requerida

```

Figure 3 - Command to launch I4Tables tool

Then, the computer or virtual machine that we want to connect to this tool must be configured. To do this, you must connect the machine to an isolated network without Internet access and configure the gateway of the network interface and the DNS server to the machine where I4Tables tool is running.

Finally, when the computer has been successfully configured and validated, I4Tables will begin to filter its connection, only allowing access to the domains specified in the rules file.

```
[deny] geover.prod.do.dsp.mp.microsoft.com -> 10.127.64.123
[deny] activation-v2.sls.microsoft.com -> 10.127.149.201
[deny] activation-v2.sls.microsoft.com -> 10.127.191.7
[allowed] incide-server.local -> 192.168.101.1
[duplicated] incide-server.local -> 192.168.101.1
[deny] v20.events.data.microsoft.com -> 10.127.222.88
```

Figure 4 - Example of allowed and denied domains

3.2.2 *Virtual Machine management*

The application developed in this project requires a tool to create and manage virtual machines, in a simple but customizable way, to simulate the execution of malware samples in the guests.

After reviewing different alternatives, the following tools have been selected:

QEMU-KVM: full virtualization system used by default on GNU/Linux systems.

Libvirt: API that allows management for different virtualization systems, including KVM.

Vagrant: tool that allows us to create and customize lightweight, reproducible, and portable environments, as well as automate the creation and management of virtual machines.

Focusing on Vagrant, the basic concepts to understand how this technology works are defined below.

- **Box:** complete and self-contained image of an operating system environment[3].
- **Vagrantfile:** Ruby file that contains the definition of the virtual machines. This includes which box Vagrant needs to run, as well as how to configure and provision these machines.

```
Vagrant.configure("2") do |config|
  config.vm.box = "PCSIM"
  config.vm.guest = :windows

  config.vm.communicator = "winrm"
  config.winrm.username = "Marc"
  config.winrm.password = "test"

  config.vm.synced_folder '.', '/vagrant', disabled: true

  config.vm.network "private_network", ip: "192.168.103.25"

  config.vm.provider :libvirt do |libvirt, override|
    libvirt.management_network_name = "network-management"
    libvirt.management_network_address = "192.168.102.0/24"
    libvirt.management_network_mode = "none"
    libvirt.nic_model_type = 'e1000'
    libvirt.memory = 4092
    libvirt.cpus = 2
  end
end
```

Figure 5 – Example of Vagrantfile

3.2.3 *Database*

Databases are a crucial part of modern applications, as they store the data that is used to feed them. Therefore, to manage all the information necessary for the operation of Tartarus we decided to use **PostgreSQL**, which is a relational database management system.

The tables generated for Tartarus will be detailed in sections *7.4 Database tables* and *8.3 Database tables*.

Finally, we also need to select a technology to interact and manipulate the database from a Python code. In this case, we have decided to use **SQLAlchemy**, which is an object relational mapping (ORM) that allows us to connect safely and easily to our database programmatically, without the need to execute SQL queries to manipulate the data.

3.2.4 *Web Framework*

Tartarus is divided into two main components: an API Rest server and a command line interface.

Focusing on the RESTful API, it allows to receive or send data to a server using HTTP requests and process certain task that are executed in the server side. For this project, we have decided to use **Flask**, which is a web framework that provides several tools, libraries and technologies to build web applications or APIs.

We have used this type of technology to create a simple but automatable communication between the user and the Flask server through a command-line interface and using the HTTP protocol.

3.2.5 *Command-Line Interface*

On the other hand, we decided to use **Python Click**[4], which is a Python package for creating command line interfaces (CLI). As mentioned before, the objective of this second component is to create a simple and intuitive way to interact with the Flask server that will process and execute the commands given by the user.

The following image shows the CLI that we have created for this project.



Figure 6 - Tartarus CLI

Furthermore, in order to send the crafted HTTP requests to the server with all the necessary data, we are going to use the **Python Requests**[5] library.

3.2.6 Monitoring and visualization of logs

Finally, an application is needed to centralize, manage, monitor and visualize events and logs from a host. We have built this part based on the idea of the project developed by Mónica Martín which implements a Security Operations Center (SOC). Therefore, the installation of this software has been followed based on her report[6].

The project mentioned above is based on the platform called **Elastic**[7], which is a collection of different open-source products. The ones that will be used for Tartarus are:

- **Elasticsearch**[8]: search and analytics NoSQL database engine. It is capable of storing, searching and analyze large volumes of data in real time.
- **Kibana**[9]: graphical interface on which the user can query and visualize the Elasticsearch database, build dashboards, generate alerts or even create reports.
- **Elastic Agent**[10]: agent that monitors and collects logs, metrics, traces, availability and other types of data from a host.
- **Fleet**: centralized management of Elastic agents. It gives you a real-time view of their status and allows to manage and update large groups of agents remotely.

4 | Methodology

4.1 Development method

To achieve the objective proposed in this project we need to implement a flexible planning method that allow us to divide the development of Tartarus system into short cycles, so that we can gradually add new functionalities while checking if they work correctly.

Therefore, the most appropriate methodology is the Lean Method[11], which is aimed to small development teams. It uses an adaptive work system instead of a predictive one. That means that the development team knows the objective but not the exact process that must be followed to achieve it. In addition, this methodology is based on short and incremental development cycles, where at the end of each cycle all the tasks completed up to that moment are evaluated.

Finally, the general objective of this methodology is to create a workflow that delivers the products with the highest possible value to the customer. To achieve this, the following principles must be met:

1. ***Remove everything unnecessary.*** Avoid everything that doesn't add value to the final product to increase the efficiency of the development process. Therefore, it is necessary to avoid unnecessary tasks or starting more than can be completed.
2. ***Principle of continuous learning.*** Constantly improve the knowledge and skills of the project members.
3. ***Quality guarantee.*** One of the goals to be achieved with this methodology is the quality of the final product. Therefore, it is of vital importance to have good initial planning, as well as constant feedback from the manager to improve the quality of the project.

4.2 Validation method

The validation of the project will be done with the project director, Carlos Fernández Sánchez, through telematic meetings on Google Meet platform.

By using a Lean work method where the development is divided into short cycles, it is important to do a meeting every week to check the progress that has been made along with the results obtained. Also, feedback will be received on possible improvements that can be implemented and possible deviations or unnecessary tasks being developed.

Finally, to check and validate whether the objectives set at the beginning of each cycle are being accomplished, we carried out different tests to verify the correct operation of the functions added to the application.

5 | Work Plan

This Master's Thesis has a course load of 12 credits. Considering that the regulations of the *Universitat Politècnica de Catalunya* indicate that each credit is equivalent to 30 hours of work, it is expected that approximately 360 hours will be dedicated to the development of the project.

This thesis started on 14th of February and is scheduled to end in June. Therefore, it is essential to have a good time planning to finish the project on the corresponding delivery date.

5.1 Tasks

This project is divided into six groups of tasks that will be detailed below.

[PM] Project Management:

- ***[PM-T1] Define the context and scope.*** One of the most relevant phases because it determines the scope and objectives of this project.
- ***[PM-T2] Define the work plan.*** The project is divided into different tasks. In addition, an estimate of the time that will be dedicated to each of these tasks is specified.
- ***[PM-T3] Follow-up meetings.*** This task includes all the meetings that take place between the director and author of this project. They are held every week with an approximate duration of half an hour.

[S] Setup:

- ***[S-T4] Prepare the work environment.*** Setup the equipment needed to develop the project, including the hardware and software that will be used to program Tartarus.

[IM] Infrastructure module:

- ***[IM-T5] Research.*** Review the different technologies that can be used to manage the infrastructure (including virtual machines, snapshots and network) of Tartarus and decide which is the one that we are going to use.
- ***[IM-T6] Implement.*** Program all the required functions of the Flask server to fulfil the objective of this module, allowing to manage the Tartarus infrastructure.
- ***[IM-T7] Test.*** Test all the implemented features and possible use cases to verify that the module works correctly.

[SM] Simulation Module:

- *[SM-T8] Research.* Plan how to automatically run malware samples of the threats on the infrastructure module.
- *[SM-T9] Implement.* Define and develop the server functionalities to allow autonomous deployment of the malware on the Tartarus infrastructure.
- *[SM-T10] Test.* Test all the implemented features and possible use cases to verify that the module works correctly.

[RM] Reporting Module:

- *[RM-T11] Research.* Investigate different ways of obtaining logs and metrics from a computer, as well as technologies that speed up and scale this process.
- *[RM-T12] Implement.* Implement and configure the technology used to collect logs and metrics of the machines hosted in the Tartarus infrastructure.
- *[RM-T13] Test.* Test all the implemented features and possible use cases to verify that the module works correctly.

[CLIM] Command-Line Interface Module:

- *[CLIM-T14] Development.* During the development of the modules detailed above, the command-line interface must also be implemented in parallel. This will be the interface that users will use to interact with the Tartarus Flask server.

[TRD] Thesis report and defense:

- *[TRD-T15] Write the thesis.* Report where the development of the project and the results obtained are detailed.
- *[TRD-T16] Defense.* The author of the project defends the thesis in front of a court that will evaluate it.

5.2 Initial Planning

The following table presents a summary of the tasks along with the approximate time that is planned to dedicate to each of them.

<i>Id</i>	<i>Task</i>	<i>Time</i>
PM	Project Management	28h
PM-T1	Define the context and scope	12h
PM-T2	Define the work plan	8h
PM-T3	Follow-up meetings	8h
S	Setup	10h
S-T4	Prepare the work environment	10h
IM	Infrastructure Module	100h

IM-T5	Research	20h
IM-T6	Implement	60h
IM-T7	Test	20h
SM	Simulation Module	120h
SM-T8	Research	20h
SM -T9	Implement	80h
SM -T10	Test	20h
RM	Reporting Module	60h
DA-T11	Research	20h
DA-T12	Implement	20h
DA-T13	Test	20h
CLIM	Command-Line Interface Module	20h
CLIM-T14	Development	20h
TD	Thesis and defense	40h
TD-T15	Write the thesis	30h
TD-T16	Defense	10h
Total		378h

Table 1 - Initial estimated time

Figure 7 shows the initial planning using a Gantt diagram.

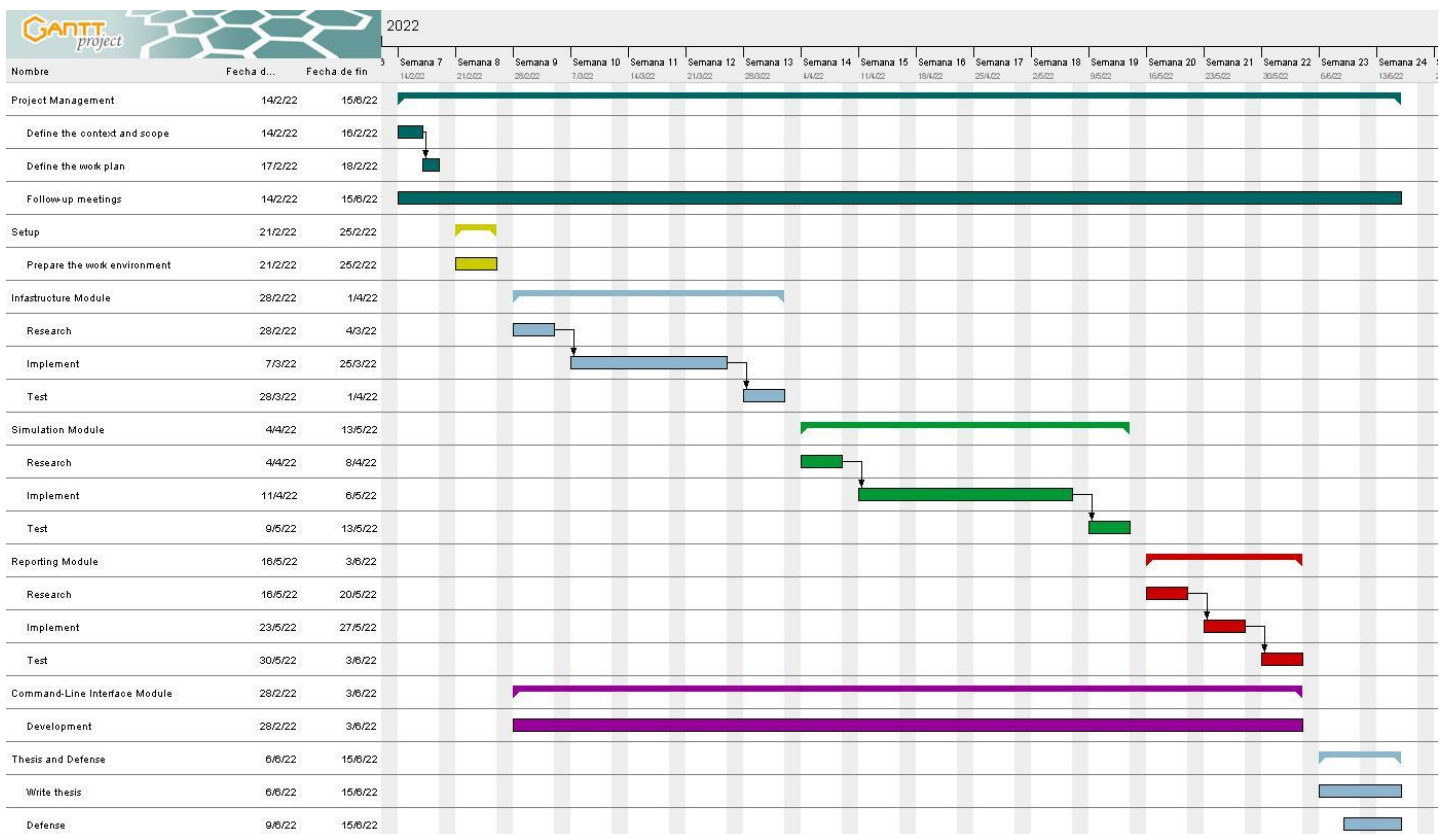


Figure 7 - Initial Gantt diagram

5.3 Final Planning

During the development of this project, an unforeseen event occurred that slightly affected the duration of some of the tasks. For this reason, the initial planning described above could not be fulfilled, increasing the final working hours.

The time needed to develop the infrastructure module was not correctly estimate. Initially, it was planned that only three weeks would be devoted to it. However, it had to be extended for another week because I couldn't finish programming all the necessary functions of this module.

Therefore, the following table presents the time that was dedicated to all the tasks of this project.

<i>Id</i>	<i>Task</i>	<i>Time</i>
PM	Project Management	28h
PM-T1	Define the context and scope	12h
PM-T2	Define the work plan	8h
PM-T3	Follow-up meetings	8h
S	Setup	10h
S-T4	Prepare the work environment	10h
IM	Infrastructure Module	120h
IM-T5	Research	20h
IM-T6	Implement	80h
IM-T7	Test	20h
SM	Simulation Module	120h
SM-T8	Research	20h
SM-T9	Implement	80h
SM-T10	Test	20h
RM	Reporting Module	60h
DA-T11	Research	20h
DA-T12	Implement	20h
DA-T13	Test	20h
CLIM	Command-Line Interface Module	20h
CLIM-T14	Development	20h
TD	Thesis and defense	40h
TD-T15	Write the thesis	30h
TD-T16	Defense	10h
Total		398h

Table 2 - Final planning

To conclude, Figure 8 shows the final planning using a Gantt diagram.

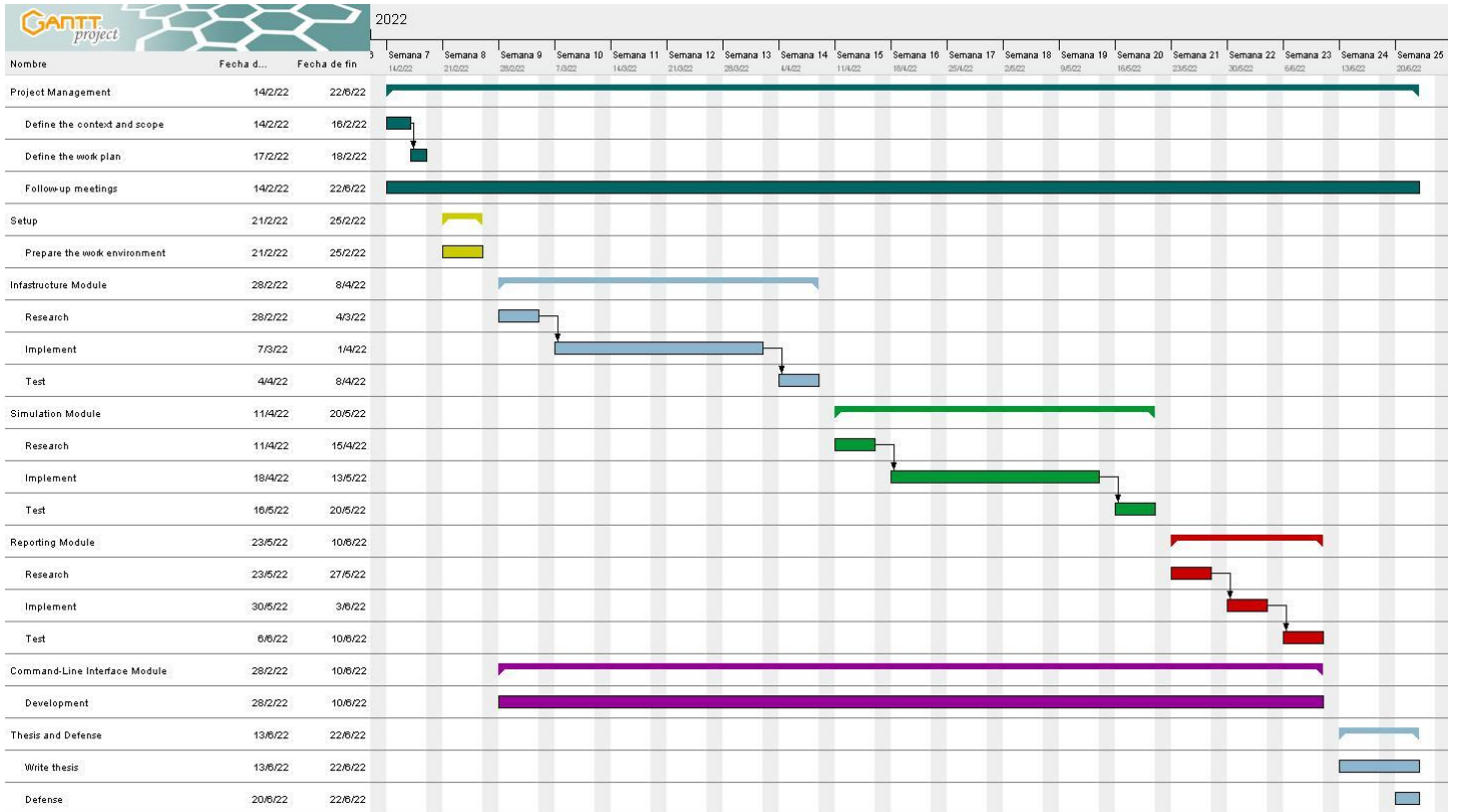


Figure 8 - Final Gantt diagram

6 | Budget

This section offers a study and analysis of the budget necessary to develop this Master's Thesis, including: human costs, generic costs, the contingency item and possible unforeseen events.

6.1 Human costs

Two roles are involved in the development of this project. On one hand, a computer engineer who is in charge of the tasks related to software development. On the other hand, a project manager who deals with the management and presentation of the work.

The following table presents the salary that has been obtained from the data offered in *indeed* website[12][13]. In addition, taxes (e.g., the contribution to Social Security) and the total cost for each profile are included.

Role	Net Salary (€/h)	Taxes (€/h)	Cost (€/h)
Software Developer	12,32	3,69	16,01
Project Manager	16,98	5,09	22,07

Table 3 - Human costs

Table 4 shows the calculation of the cost per activity (CPA) of this project.

Task	Time (hours-person)	Project Manager (€)	Software Developer (€)
Project Management			
Define the context and scope	12	264,84	0
Define the work plan	8	176,56	0
Follow-up meetings	8	176,56	0
Setup			
Prepare the work environment	10	0	160,10
Infrastructure Module			
Research	20	0	320,20
Implement	80	0	1.280,80
Test	20	0	320,20
Simulation Module			
Research	20	0	320,20
Implement	80	0	1.280,80
Test	20	0	320,20
Reporting Module			
Research	20	0	320,20
Implement	20	0	320,20
Test	20	0	320,20
Command-Line Interface Module			

Development	20	0	320,20
Thesis and defense			
Write the thesis	30	0	480,30
Defense	10	0	160,10
Total Role	0	617,96	5.923,7
Total CPA		6.541,66€	

Table 4 - Cost per activity

6.2 Generic costs

In this section the indirect costs will be define, that is, those not attributable to perform the tasks of the project.

6.2.1 Hardware

It is required a laptop to develop Tartarus application. Specifically, a Mountain Onyx 15 laptop has been used, which has an approximate lifetime of 7 years and a cost of 1599€[14].

To calculate the amortization of this product we have divided the price of the laptop by the months of useful life and then we have multiplied it by the months that the project will last.

6.2.2 Software

The software used to implement Tartarus is free or self-programmed. Therefore, this category will not have an impact on the cost.

Product	Price (€)	Vida útil (años)	Amortización (€)	Coste (€)
Laptop (Mountain Onyx 15)	1.599	7	95,18	95,18
Software	0	-	0	0
Total GC			95,18€	

Table 5 - Generic cost

6.3 Contingency

It is possible that during the development of the project may occur different obstacles or unforeseen events that negatively affect the work plan detailed above. Therefore, it is important to prevent these situations by adding an extra budget as a safety margin.

In this project, 15% of the direct and indirect costs will be added as an extra budget, resulting in:

$$\text{Contingency} = (\text{total CPA} + \text{total GC}) * 0.15 = (6.541,66 + 95,18) * 0.15 = \mathbf{995,526€}$$

6.4 Total cost

To conclude, Table 6 presents the total cost of the project.

Concept	Cost (€)
Total CPA	6.541,66
Total GC	95.18
Contingency	995,526
Total	7.632,36€

Table 6 - Total cost

7 | Project development

Throughout this section we will detail the design of the semi-automated threat analysis system implemented in this project.

7.1 Modules

In order to develop an application easily scalable, we have divided the tool into four different interconnected modules, each one with different functionalities.

In the future, if we want to add more features, we will just need to create another module.

7.1.1 Module #1 - Infrastructure

The objective of this module is to manage the infrastructure where the threat simulations will take place. This includes the disk images that corresponds to the virtualized clients' workstations, the sandboxed virtual machines and snapshots. Moreover, with this module we can also control the network and the I4Tables tool to isolate the guests.

7.1.2 Module #2 - Simulation

This second module focuses on the reproduction of the imminent threats in the virtualized workstations. To do this, a sample of the malware is uploaded into the Tartarus server and a custom agent installed in the virtual machines is in charge of obtaining and executing it.

7.1.3 Module #3 - Reporting

The reporting module focuses on the monitoring and visualization of the logs and metrics generated in the virtual machines during the execution of the threat. Thanks to this module, the user can learn the behavior of the computer during threat simulation and observe the detection and blocking capacity of a host against it.

7.1.4 Module #4 - Command-Line Interface (CLI)

The last module consists of the command-line interface that allows the user to interact with the rest of the modules.

7.2 Infrastructure of the application

In the following figure we can observe an overview of Tartarus architecture.

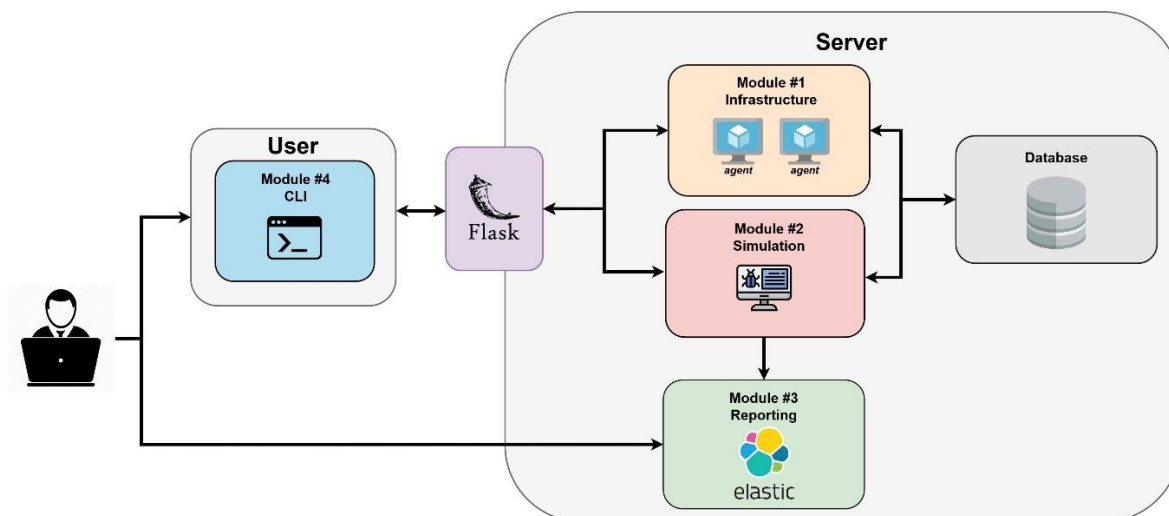


Figure 9 - Tartarus architecture

As shown in Figure 9, the architecture is divided into two main parts. On the one hand, the **user's computer** where the command-line interface is located. On the other hand, the **server** where the Tartarus application (including infrastructure, simulation and reporting modules) is running together with the PostgreSQL database.

7.2.1 Components

Below are detailed the most important components of Tartarus:

- **Flask server:** API responsible of processing and executing the actions that the user requests through the CLI. This server includes two different modules (infrastructure and simulation) that communicate with the Elastic tool and PostgreSQL database.
- **Command-Line Interface:** allows the user to communicate in a simple and centralized way with the Flask server. This CLI will generate the HTTP requests based on the commands specified by the user and send them to the server.
- **Agent:** small program programmed in python that is installed in every virtual machine managed by Tartarus, allowing to control the system where it is installed. The objective of this agent is to constantly ask the server if he has any threat simulation to execute, and in the case that he receives a simulation, this agent will download the payload in the virtual machine and execute it.
- **Elastic:** set of tools for data ingestion, storage, analysis and visualization that make up the third module (reporting). The objective of this component is to understand the functioning of the threats and observe the detection and blocking

capacity of the system against them using the logs of the machines generated during the threat simulation.

- **Database:** PostgreSQL database where Tartarus can store and manage the necessary data for the proper functioning of the application.

7.3 Network

The management of the network is one of the most critical components of Tartarus, since if we want to safely run malware in the virtual machines, they must be properly isolated to not compromise any of the systems that are on the network or make unwanted communications.

To solve this problem and filter all the communications that come from the machines where the malware is executed, we have implemented two virtual networks and created a virtual machine where the I4Tables tool (*2.2.1 Isolated Local Network*) is running.

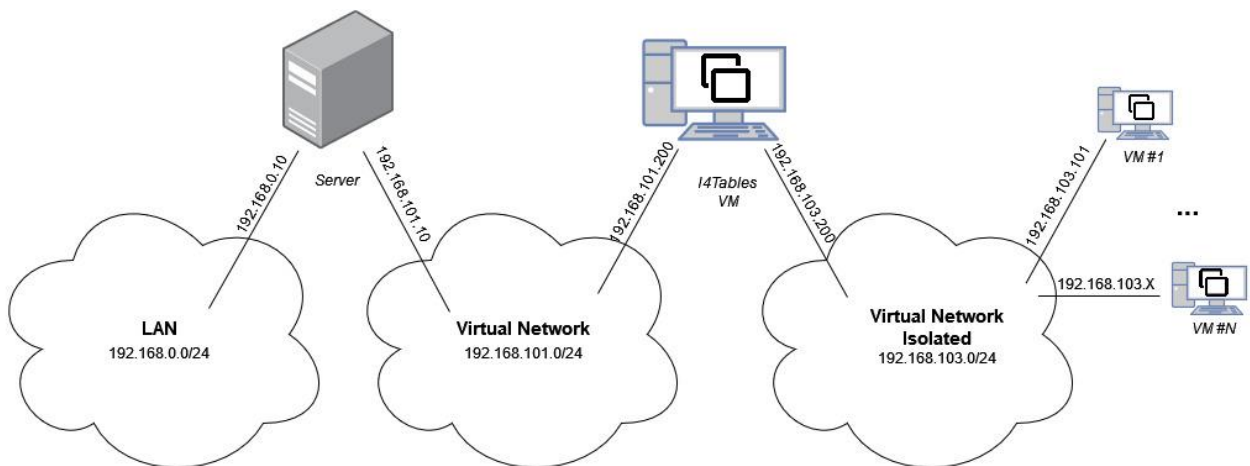


Figure 10 – Overview of Tartarus network

As shown in Figure 10, the implemented solution consists of:

- First, the server where the Tartarus application is running is connected to the LAN (Local Area Network) and it has full internet access.
- Secondly, we have created a virtual network (*192.168.101.0/24*) with NAT and DHCP enabled to give outbound network access. In this virtual network there are connected two different hosts: the server with Tartarus and the virtual machine with the I4Tables tool.
- Finally, we have created an isolated virtual network (*192.168.103.0/24*), which doesn't have NAT and DHCP enabled. The guests that are connected to this network will send all the traffic through the I4Tables machine, which will only allow access to the domains and subdomains configured in the rule's file.

In this second virtual network is connected the I4Tables virtual machine and all the guests where we want to deploy malicious samples.

8 | Module #1 - Infrastructure

8.1 Introduction

The objective of this module is to manage the infrastructure of Tartarus (including disk images, virtual machines and snapshots) where the threat simulations will be executed. In addition, we will also be able to manage the network and the I4Tables application to isolate the guests.

As mentioned before, we are going to use the combination of QEMU-KVM (virtualization system on GNU/Linux systems) and Vagrant (tool that allows us to automate the creation and management of virtual machines).

8.2 Components

We can differentiate four different components with this module:

- **Virtualized Disk Images:** virtual hard disk image extracted from the physical disk of a clients' workstation. The formats currently supported by Tartarus are VHDX and QCOW2. Once they have been processed by Tartarus, they are converted into Vagrant boxes.
- **Virtual Machines:** Vagrant virtual machines that are created using the virtualized disk images.
- **Snapshots:** snapshot of the state of a system at a given time. The objective is that once the virtual machine has executed a threat, it can return to the original state prior to the execution of the malware.
- **Network:** includes all the components needed to manage the network in order to isolate the virtual machines. That is the network virtual machine, the I4Tables tool itself and the management of I4Tables profiles.

8.3 Functionalities

The following use case diagram shows the actions grouped by categories that can be performed on Tartarus regarding the infrastructure module.

In the next sections we will explain each of these categories in detail.

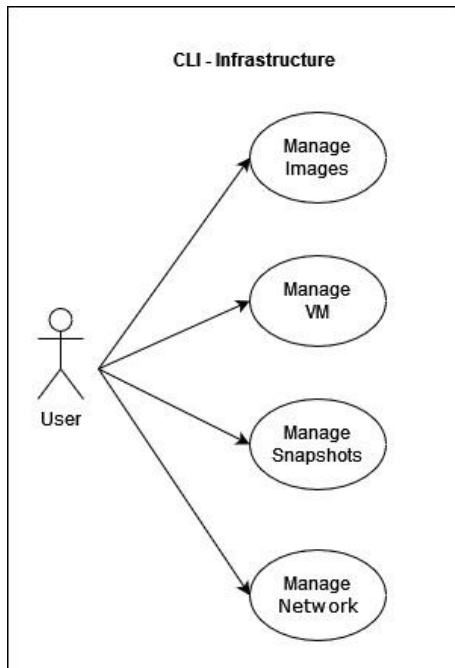


Figure 11 - Infrastructure use case diagram

8.4 Database tables

We have defined the following tables in Postgres SQL database to store all the necessary information for the correct functioning of the infrastructure module.

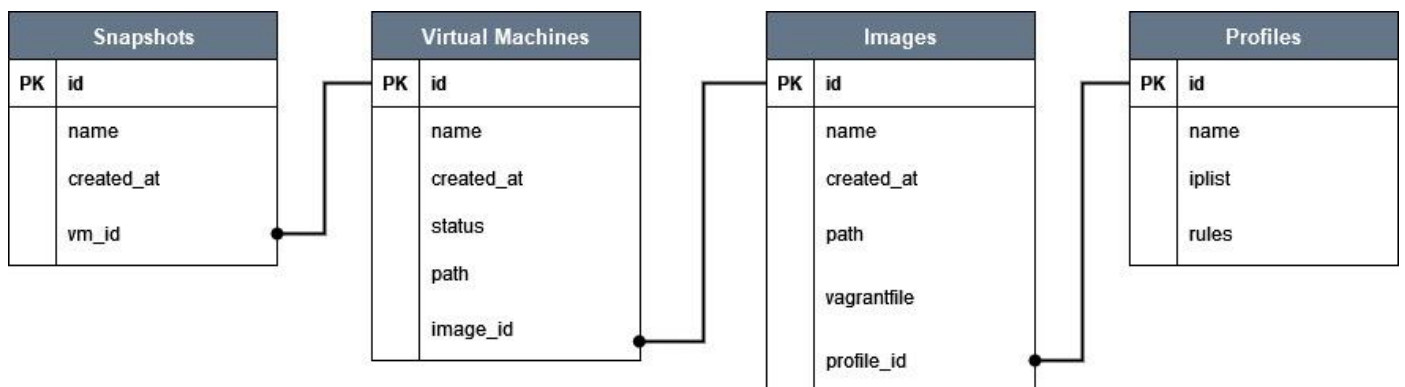


Figure 12 - Infrastructure DB Tables

8.5 Previous configuration

Before the user can use this module and start managing the infrastructure of Tartarus, he needs to prepare the clients' workstations where the threat simulations will be performed.

8.5.1 Enable WinRM

Windows Remote Management (WinRM) is a Windows built-in remote management protocol. It allows to invoke scripts and execute PowerShell commands on remote Windows machines without having to use Remote Desktop Protocol (RDP) or log into the remote machine.

Vagrant uses this protocol to provision and configure the Windows virtual machines. Consequently, it is important that this service is enabled before creating the virtual disk image from the computer. To start this service, we just need to run the following command:

```
> Enable-PSRemoting -Force
```

8.5.2 Prepare Vagrantfile

Before importing an image into Tartarus, we need to prepare the Vagrantfile with the description the virtual machine, the resources that we want to allocate and how we want to configure and provision the machine.

8.5.3 Extract disk image

Finally, the physical computer must be converted into a virtual disk image. In order to do so, we can use the Sysinternal tool from Windows called *disk2vhd*[16], which is a utility that creates VHD (Virtual Hard Disk) versions of physical disks.

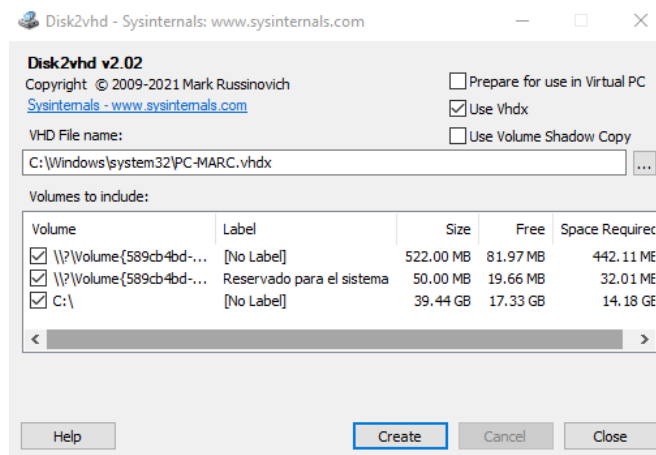


Figure 13 - Disk2vhd interface

8.6 Images

Images are one of the main components of Tartarus. As mentioned above, this represents a virtualized disk image that has been extracted from a physical computer.

The operations that can be performed in Tartarus with this component can be seen in the following use case diagram:

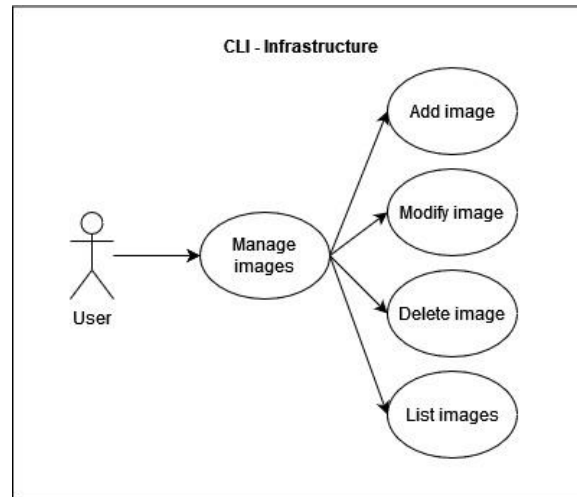


Figure 14 - Images use case diagram

8.6.1 Add image

Description:

This command imports a new disk image into Tartarus. The server converts the disk image that the users has specified in the *path* option into a Vagrant box [17]. Once it has been converted, Tartarus adds the box into Vagrant.

Finally, if no fault occurs during this process, a new entry is created in the *image* table from the database.

Command:

```
tartarus >> infrastructure images add --format <> --name <> --path <> ( --profile <> )
```

Options:

- *--format*: format of the disk image. Currently, it supports *qcow2* and *vhdx*.
- *--name*: identifier name that will be assigned to the image.
- *--path*: path where the disk image is located.
- *--profile*: [Optional] i4tables profile. If no specified, it assigns the default profile.

Example:

```
tartarus >> infrastructure images add --format QCOW2 --name PCW10 --path /home/rbarcelo/PC.qcow2
Image has been imported.
```

Figure 15 - Add image

8.6.2 *Modify image*

Description:

The user has the possibility to modify the profile or the Vagrantfile linked to a disk image.

Command:

```
tartarus >> infrastructure images modify --image <>
--option [profile --profile // vagrantfile --vagrantfile]
```

Options:

- *--image*: name of the image that the user wants to modify.
- *--option*: which parameter of an image the user wants to modify. Currently, the profile and vagrantfile can be changed.
 - *--profile*: path of the new profile. Only required in case the option is equal to profile.
 - *--vagrantfile*: path of the new vagrantfile. Only required in case the option is equal to vagrantfile.

Example:

```
tartarus >> infrastructure images modify --image PCW10 --option Vagrantfile
--vagrantfile /home/rbarcelo/W10.vagrantfile
Image has been updated.
```

Figure 16 - Modify image

8.6.3 *Delete image*

Description:

The user can delete a disk image from Tartarus. The server will remove the box from Vagrant and all the files related to the image. Finally, the database entry is also deleted.

Command:

```
tartarus >> infrastructure images delete --image <> --force
```

Options:

- *--image*: name of the image that will be deleted.
- *--force*: runs the command without asking for user confirmation.

Example:

```
tartarus >> infrastructure images delete --name PCW10 --force
Image has been removed.
```

Figure 17 - Delete image

8.6.4 *List images*

Description:

This command returns a list with all the disk images that are imported into Tartarus.

Command:

```
tartarus >> infrastructure images list
```

Example:

```
tartarus >> infrastructure images list
```

ID	Name	Vagrantfile	Created At	Path Box	Profile ID
7	PCSIM	default	Thu, 19 May 2022 19:25:22 GMT	/home/rbarcelo/Documentos/TFM/files/PCSIM.box	3
10	PCW10	PCW10.vagrantfile	Wed, 01 Jun 2022 16:45:04 GMT	/home/rbarcelo/Documentos/TFM/files/PCW10.box	3

Figure 18 - List images

8.7 Virtual Machines

Once we have imported the virtualized disk image into Tartarus, we can now easily create and manage virtual machines taking advantage of Vagrant features.

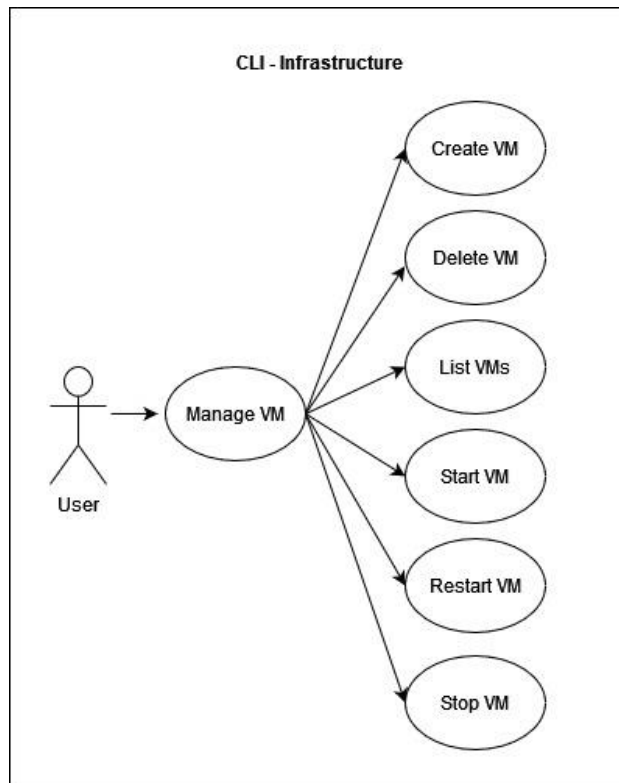


Figure 19 - VM use case diagram

8.7.1 Create VM

Description:

This command creates a new virtual machine from an image that has been imported into Tartarus. The server internally creates a new folder and adds the necessary configuration files (including the Vagrantfile) to setup and initialize the Vagrant environment.

If no fault occurs during this process, a new entry is created in the *VirtualMachines* table from the database.

Command:

```
tartarus >> infrastructure vm create --image <> --name <>
```

Options:

- *--image*: name of the image that is going to be used for the virtual machine.
- *--name*: identifier name that will be assigned to the virtual machine.

Example:

```
tartarus >> infrastructure vm create --image PCW10 --name PCW10-1  
VM has been created.
```

Figure 20 - Create virtual machine

8.7.2 Delete VM

Description:

It deletes a virtual machine from Tartarus. The server destroys all the resources that were created during the provision of the machine. Finally, the database entry is deleted.

Command:

```
tartarus >> infrastructure vm delete --vm <> --force
```

Options:

- *--vm*: name of the virtual machine that will be deleted.
- *--force*: runs the command without asking for user confirmation.

Example:

```
tartarus >> infrastructure vm delete --name PCW10-1 --force  
VM has been removed.
```

Figure 21 - Delete virtual machine

8.7.3 List VMs

Description:

This command lists all the virtual machines that are currently created in Tartarus.

Command:

```
tartarus >> infrastructure vm list
```

Example:

```
tartarus >> infrastructure vm list
| ID | Name | State | Created At | Path | IMG-ID |
|----|-----|-----|-----|-----|-----|
| 54 | PCSIM-3 | Stopped | Thu, 26 May 2022 10:14:05 GMT | /home/rbarcelo/Documentos/TFM/vm/PCSIM/PCSIM-3 | 7 |
| 60 | PCW10-1 | New | Wed, 01 Jun 2022 17:00:51 GMT | /home/rbarcelo/Documentos/TFM/vm/PCW10/PCW10-1 | 11 |
```

Figure 22 - List virtual machines

8.7.4 Start VM

Description:

The user can start a virtual machine using this command. Internally, the Tartarus server configures and provision the VM based on the Vagrantfile.

Once this process has finished, the user will be able to connect via GUI or terminal and interact with the guest.

Command:

```
tartarus >> infrastructure vm start --name <>
```

Options:

- `--name`: name of the virtual machine.

Example:

```
tartarus >> infrastructure vm start --name PCW10-1
VM has been started.
```

Figure 23 - Start virtual machine

8.7.5 Stop VM

Description:

This command gracefully shuts down a powered on virtual machine by running the guest OS shutdown mechanism. In case this process fails, Vagrant will just shut off the power of the machine.

Command:

```
tartarus >> infrastructure vm stop --name <>
```

Options:

- *--name*: name of the virtual machine.

Example:

```
tartarus >> infrastructure vm stop --name PCW10-1  
VM has been stopped.
```

Figure 24 - Stop virtual machine

8.7.6 Restart VM

Description:

This is the equivalent of running a stop followed by starting again the virtual machine

Command:

```
tartarus >> infrastructure vm restart --name <>
```

Options:

- *--name*: name of the virtual machine.

Example:

```
tartarus >> infrastructure vm restart --name PCW10-1  
VM has been restarted.
```

Figure 25 - Restart virtual machine

8.8 Snapshots

Snapshots capture the state and data of virtual machines, allowing to restore them to the previous state at which the snapshot was captured.

This component is of great importance, as the malware samples that will be executed during a threat simulation can damage the virtual machines to a practically unusable condition. Therefore, the state before the attack simulation must be restored.

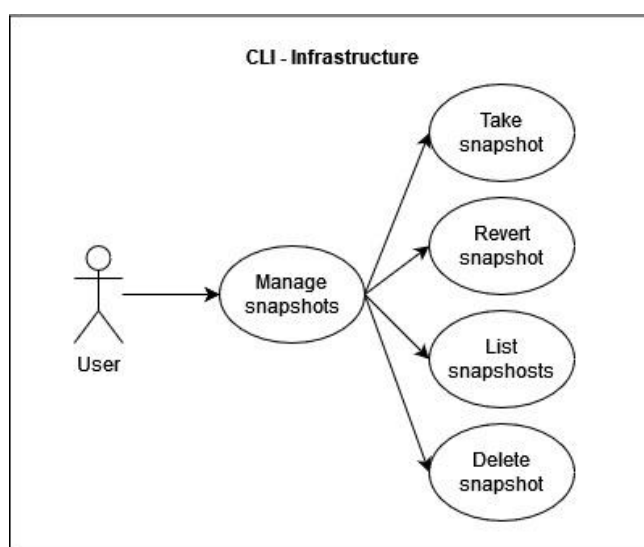


Figure 26 - Snapshots use case diagram

8.8.1 Take snapshot

Description:

This command takes a snapshot of the virtual machine specified by the user. In addition, if no fault occurs during this process, a new entry is created in the *Snapshots* table from the database.

Command:

```
tartarus >> infrastructure snapshots take --vm <> --name <>
```

Options:

- *--name*: identifier name that will be assigned to the snapshot.
- *--vm*: name of the virtual machine.

Example:

```
tartarus >> infrastructure snapshots take --vm PCW10-1 --name PCW10-Base  
Snapshot has been created.
```

Figure 27 - Take snapshot

8.8.2 Revert snapshot

Description:

This command restores the virtual machine to the previous state at which the snapshot was taken.

Command:

```
tartarus >> infrastructure snapshots revert --name <> --force
```

Options:

- `--name`: name of the snapshot that will be reverted.
- `--force`: runs the command without asking for user confirmation.

Example:

```
tartarus >> infrastructure snapshots revert --name PCW10-Base --force  
Snapshot has been reverted.
```

Figure 28 - Revert snapshot

8.8.3 List snapshots

Description:

Lists all the snapshots that are available in Tartarus.

Command:

```
tartarus >> infrastructure snapshots list
```

Example:

```
tartarus >> infrastructure snapshots list  
| ID | Name | Status VM | Created At | VM ID |  
|----|-----|-----|-----|-----|  
| 17 | PCW10-Base | Stopped | Wed, 01 Jun 2022 17:17:02 GMT | 60 |
```

Figure 29 - List snapshots

8.8.4 Delete snapshot

Description:

The user can delete a snapshot associated with a virtual machine. Moreover, the database entry that corresponds to the snapshot is removed.

Command:

```
tartarus >> infrastructure snapshots delete --name <> --force
```

Options:

- `--name`: name of the snapshot.
- `--force`: runs the command without asking for user confirmation.

Example:

```
tartarus >> infrastructure snapshots delete --name PCW10-Base --force  
Snapshot has been deleted.
```

Figure 30 - Delete snapshot

8.9 Network

This section includes all the components needed in Tartarus to manage the network that will be used to isolate the virtual machines and execute the malware samples safely without compromising other hosts. Therefore, it includes:

- Management of the virtual machine where I4Tables tool is located.
- Management of the I4Tables tool itself.
- Management of I4Tables profiles (rules and iplist configuration files).

The operations that can be performed in Tartarus with this component can be seen in the following use case diagram:

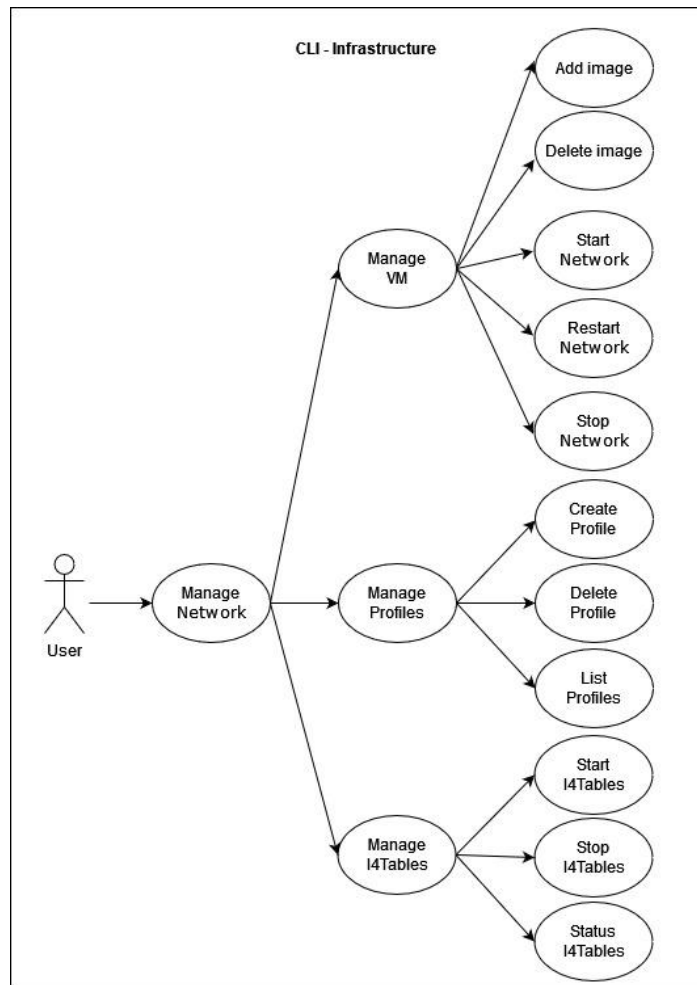


Figure 31 - Network use case diagram

8.9.1 Manage VM - Add image

Description:

This command imports a disk image into Tartarus that will be used only for managing the network. Then, creates a new virtual machine based on this image, adding the necessary configuration files to setup and initialize the Vagrant environment.

In case that the image is already imported, this command will only create the virtual machine.

Command:

```
tartarus >> infrastructure network vm add
```

Example:

```
tartarus >> infrastructure network vm add  
Image for I4tables successfully imported.
```

Figure 32 - Add network image

8.9.2 Manage VM - Delete image

Description:

The user can decide if he wants to delete only the virtual machine or also the virtualized disk image.

Command:

```
tartarus >> infrastructure network vm delete --option [all // vm] --force
```

Example:

```
tartarus >> infrastructure network vm delete --option all --force  
I4tables VM and IMG deleted.
```

Figure 33 - Delete network image

8.9.3 Manage VM - Start network

Description:

This command starts the network virtual machine where I4Tables tool is located. The server configures and provision the VM based on the Vagrantfile. Once this process has finished, the user will be able to connect via GUI or terminal and interact with the guest.

Command:

```
tartarus >> infrastructure network vm start
```


Example:

```
tartarus >> infrastructure network vm start
I4tables VM started.
```

Figure 34 - Start network virtual machine

8.9.4 Manage VM - Stop network

Description:

This command shuts down the network virtual machine by running the guest OS shutdown mechanism. In case this fails, Vagrant will just shut off the power of the machine.

Command:

```
tartarus >> infrastructure network vm stop
```

Example:

```
tartarus >> infrastructure network vm stop
I4tables VM stopped.
```

Figure 35 - Stop network virtual machine

8.9.5 Manage VM - Restart network

Description:

This is the equivalent of running a stop followed by starting again the network virtual machine.

Command:

```
tartarus >> infrastructure network vm restart
```

Example:

```
tartarus >> infrastructure network vm restart
I4tables VM restarted.
```

Figure 36 - Restart network virtual machine

8.9.6 Manage profiles - Create profile

Description:

It creates a new profile for the I4Tables tool. During this process, the server will save the rules and iplist files that the user has specified in the command.

Finally, if no fault occurs during this process, a new entry is created in the *Profiles* table from the database.

Command:

```
tartarus >> infrastructure network profile create --name <> --rules <> --iplist <>
```

Options:

- *--name*: identifier name that will be assigned to the profile.
- *--rules*: path where “rules” file is located.
- *--iplist*: path where “iplist” file is located.

Example:

```
tartarus >> infrastructure network profiles create --name UPC --rules /home/rbarcelo/rules.txt --iplist /home/rbarcelo/iplist.txt  
Profile has been created.
```

Figure 37 - Create profile

8.9.7 Manage profiles - Delete profile

Description:

This command deletes the profile specified by the user and all the files associated with it. Moreover, the corresponding database entry is removed.

Command:

```
tartarus >> infrastructure network profiles delete --name <> --force
```

Options:

- *--name*: name of the profile that will be delete.
- *--force*: runs the command without asking for user confirmation.

Example:

```
tartarus >> infrastructure network profiles delete --name UPC --force  
Profile has been deleted.
```

Figure 38 - Delete profile

8.9.8 Manage profiles - List profiles

Description:

Lists all the profiles that are available in Tartarus.

Command:

```
tartarus >> infrastructure network profiles list
```

Example:

```
tartarus >> infrastructure network profiles list
| ID | Name |
|----|-----|
| 3 | default |
| 5 | lnk-sample |
| 6 | internet |
| 7 | emotet |
| 8 | conti-xml |
| 9 | UPC |
```

Figure 39 - List profiles

8.9.9 Manage I4Tables - Start i4tables

Description:

This command starts the i4tables tool using the profile specified by the user.

Command:

```
tartarus >> infrastructure network i4tables start --profile <>
```

Options:

- *--profile*: name of the profile that will be used to initialize i4tables tool.

Example:

```
tartarus >> infrastructure network i4tables start --profile UPC
I4tables started.
```

Figure 40 - Start I4Tables tool

8.9.10 Manage I4Tables - Stop i4tables

Description:

This command stops the i4tables process.

Command:

```
tartarus >> infrastructure network i4tables stop
```

Example:

```
tartarus >> infrastructure network i4tables stop
I4tables stopped.
```

Figure 41- Stop I4Tables tool

8.9.11 Manage I4Tables - Status i4tables

Description:

Returns the status of the i4tables tool. It informs which of the components of the tool is missing or failing.

Command:

```
tartarus >> infrastructure network i4tables status
```

Example:

```
tartarus >> infrastructure network i4tables status  
I4tables is not running: HTTP server and I4Tables process are missing
```

Figure 42 - Status I4Tables tool

9 | Module #2 - Simulation

9.1 Introduction

The objective of this second module is the reproduction of imminent threats in the virtual machines that corresponds to a client's workstation. These threats include techniques that are currently being used by APT groups or cybercriminals, zero-day exploits, malware samples, among others.

In addition, the objective is to automatically perform the deployment and subsequently execution of the malware so that the user only needs to insert the details of the threat.

9.2 Components

We can differentiate two different components used in this module:

- **Tasks:** description of a threat simulation. It includes the commands along with the payloads that will be executed.
- **Agent:** small piece of code installed on the virtual machines. This agent allows the Tartarus server to take control of the machines where it is installed and manage the execution of the tasks.

9.3 Database tables

We have defined the following tables in Postgres SQL to store all the necessary information for the correct functioning of the simulation module.

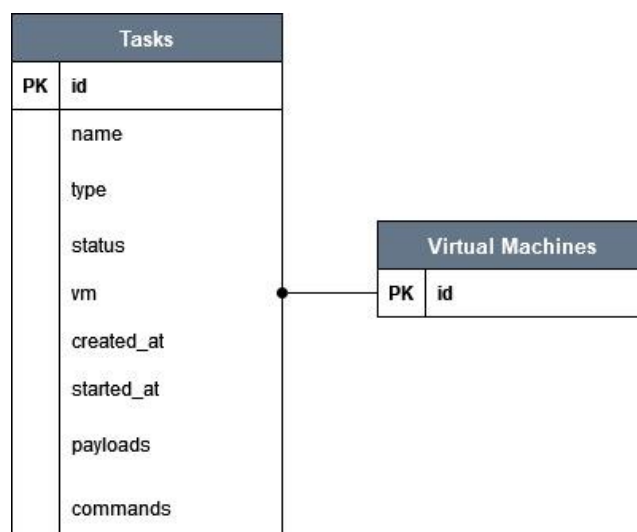


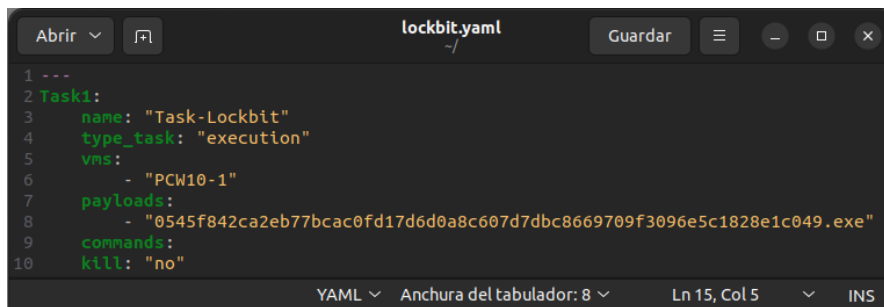
Figure 43 - Simulation DB Tables

9.4 Tasks

Tasks, as mentioned above, consists of everything necessary to execute a threat in a virtual machine. The most relevant parts are:

- **Payloads:** malware sample of the threat that will be simulated.
- **Commands:** shell commands that specify how to execute the payloads. This component is not always necessary, only when the sample requires it.

The following image presents an example of a YAML file where a task is specified with all the required parameters.



```
1 ---
2 Task1:
3   name: "Task-Lockbit"
4   type_task: "execution"
5   vms:
6     - "PCW10-1"
7   payloads:
8     - "0545f842ca2eb77bcac0fd17d6d0a8c607d7dbc8669709f3096e5c1828e1c049.exe"
9   commands:
10  kill: "no"
```

Figure 44 - Task specification

The following picture presents all the possible operations that the user can do in Tartarus regarding the tasks.

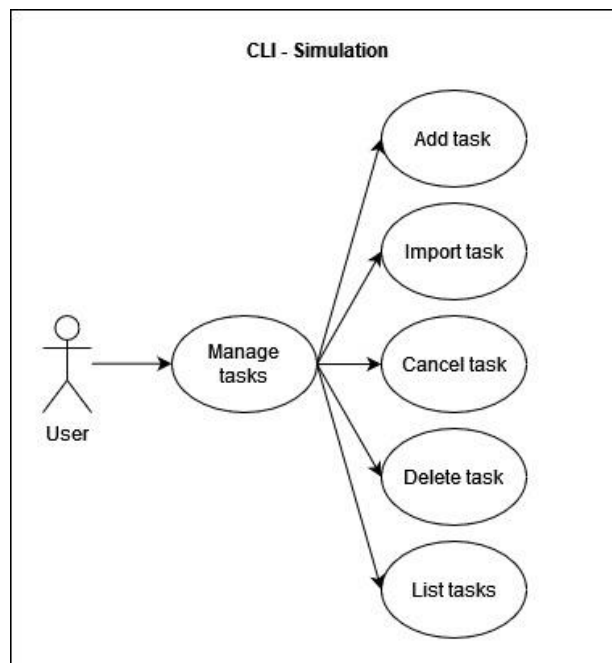


Figure 45 - Tasks use case diagram

9.4.1 Add task

Description:

This command adds a new threat simulation to Tartarus. Therefore, the malicious files specified in the *payload* argument are uploaded to the server and a new entry in the *Tasks* table is created with the details specified by the analyst.

Command:

```
tartarus >> simulation tasks add --name <> --task <> --vm <>
--payloads <> --commands <> --kill <>
```

Options:

- *--name*: identifier name that will be assigned to the task.
- *--task*: type of task. Currently it only supports execution tasks.
- *--vm*: list of virtual machines where the task will be executed.
- *--commands*: [Optional] commands that the user wants to execute.
- *--payloads*: malware files that the user wants to execute.
- *--kill*: kill the malware process that is running after 30 seconds.

Example:

```
tartarus >> simulation tasks add --name Conti --task execution --vm '["PCW10-1"]'
--payload '["/home/rbarcelo/conti-sample.xml"]' --command '['] --kill no
Task has been scheduled.
```

Figure 46 - Add task

9.4.2 Import task

Description:

The user can automatically create multiple tasks at once. To do so, he needs to specify all the tasks in a YAML file with all the required parameters. The server uploads the malicious files stored in *path* and creates the corresponding entries in the *Tasks* table.

Command:

```
tartarus >> simulation tasks import --yaml <> --path <>
```

Options:

- *--yaml*: path of the YAML file with the tasks that the user wants to create.
- *--path*: path of the directory where the payloads are stored.

Example:

```
tartarus >> simulation tasks import --yaml /home/rbarcelo/conti_xml.yaml
--path /home/rbarcelo/conti_xml/
Task has been imported and scheduled.
```

Figure 47 - Import task

9.4.3 Cancel task

Description:

This command cancels the execution of a task. If there are virtual machines running the payloads of the cancelled threat simulation, the agent will try to kill the process.

Command:

```
tartarus >> simulation tasks cancel --name <> --force
```

Options:

- *--name*: name of the task that will be cancelled.
- *--force*: runs the command without asking for user confirmation.

Example:

```
tartarus >> simulation tasks cancel --name Conti-1 --force  
Task has been canceled.
```

Figure 48 - Cancel task

9.4.4 Delete task

Description:

This command deletes a task, removing the payloads that have been uploaded to the server and the corresponding database entry.

Command:

```
tartarus >> simulation tasks delete --name <> --force
```

Options:

- *--name*: identifier name of the task.
- *--force*: runs the command without asking for user confirmation.

Example:

```
tartarus >> simulation tasks delete --name Conti-1 --force  
Task has been removed.
```

Figure 49 - Delete task

9.4.5 List tasks

Description:

This command lists all the tasks that are currently created in Tartarus.

Command:

```
tartarus >> simulation tasks list
```


Example:

```
tartarus >> simulation tasks list
```

ID	Name	VM	Type task	Status	Created At	Started At	Payloads	Command	Kill
209	Conti-1	PCW10-1	execution	remaining	Mon, 30 May 2022 21:31:54 GMT		['conti-sample.xml']	[]	no

Figure 50 - List tasks

9.5 Agent

The agent is a small piece of code written in Python that is installed in the virtual machines managed by the infrastructure module of Tartarus. It allows the server to take control of the guests and execute the payloads of the threat simulations that are assigned to the virtual machine that the agent is controlling.

Operation

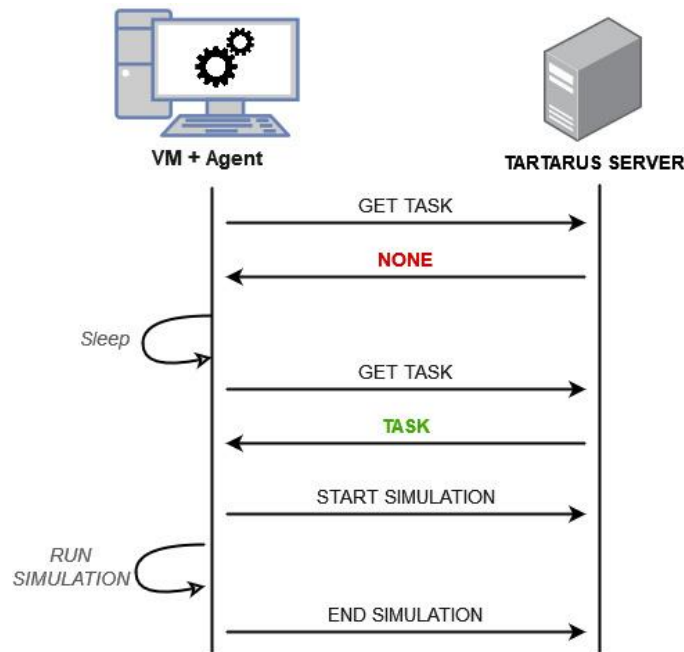


Figure 51 - Operation of Tartarus agent

Regarding the operation of the Tartarus agent, it will constantly request to the Flask server if he has any pending task.

Once the user assigns a task to the virtual machine that the agent is controlling, it will receive all the information of the threat simulation and download the malicious payloads. Then, the simulation process starts. If the user has not specified any command, the files will be executed directly. Otherwise, they will be executed using the specified commands.

Finally, when the simulation process has finished, the agent notifies to the server and start asking again for a new task.

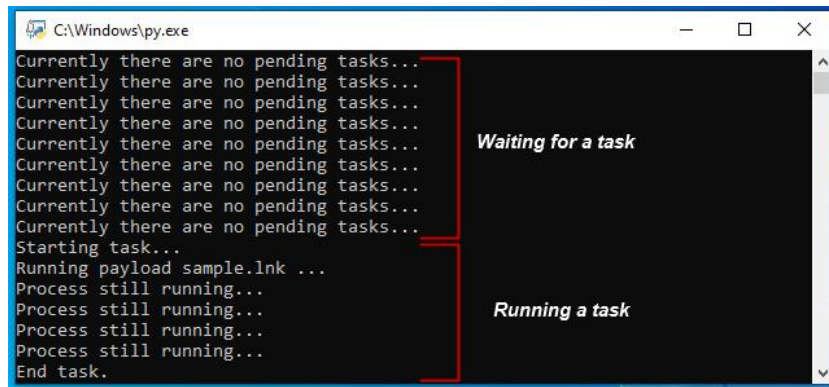


Figure 52 - Example of the operation of an agent

Use case diagram

The following diagram shows a summary of the possible actions. In this case, the interaction with this component is between the agent installed in the virtual machines and the Tartarus server.

In addition, it should be noted that the agents are independent of each other. Therefore, these actions can be performed on several agents at the same time. This fact allows to execute the tasks in parallel on different virtual machines, considerably reducing the time required to perform the threat simulations.

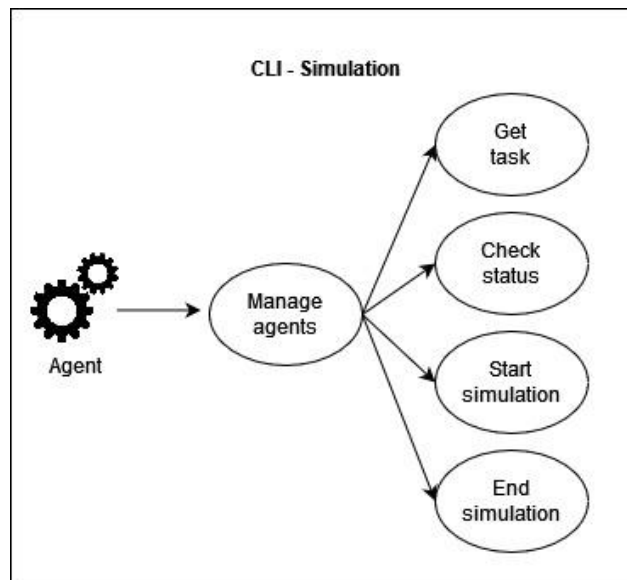


Figure 53 - Agents use case diagram

9.5.1 Get task

Description:

The agent requests to the server if he has any pending tasks assign to it. If so, the server sends all the information related to the task that is stored in the database, including the malicious payloads.

9.5.2 Start simulation

Description:

When the agent receives a new task (through 8.5.1 *Get task*), it starts to configure and execute the threat simulation.

Here we can occur two different cases: if the analyst has not specified any command in the task, the files will be executed directly. Otherwise, they will be executed using the specified commands.

9.5.3 Check status

Description:

While the agent is running the payloads of the threat simulation, it will constantly ask the server for the status of the task. In case that the analyst cancels it, the agent will try to kill the malware sample that is currently running and stop the threat simulation.

9.5.4 End simulation

Description:

Once the agent has finished executing the task or has reached the timeout, it informs to the server that the threat simulation has finished. From this moment on, the agent starts again the process asking for a new task (8.5.1 *Get task*).

10 | Module #3 – Reporting

10.1 Introduction

The objective of this third module is to observe the detection, blocking capacity and behavior of a workstation and the security policies configured on it against a threat.

We have configured a system to monitor and visualize the logs and metrics generated in the virtual machine during the execution of the threat simulation. As mentioned in section 2.4.3 *Monitoring and visualization of logs*, the implemented solution is based on Elastic tool. Specifically, the following components are used: Elasticsearch, Kibana, Elastic Agent and Fleet.

10.2 Fleet

First of all, we need to configure the Elastic Agents that will be installed in the virtual machines managed by Tartarus in order to monitor them.

We need to access to Kibana web interface and create a Fleet policy, which is a collection of inputs and settings that defines the data that is going to be collected by the agent. In addition, Elastic provides a simple way to collect data from popular apps, services or systems called Integrations[18].

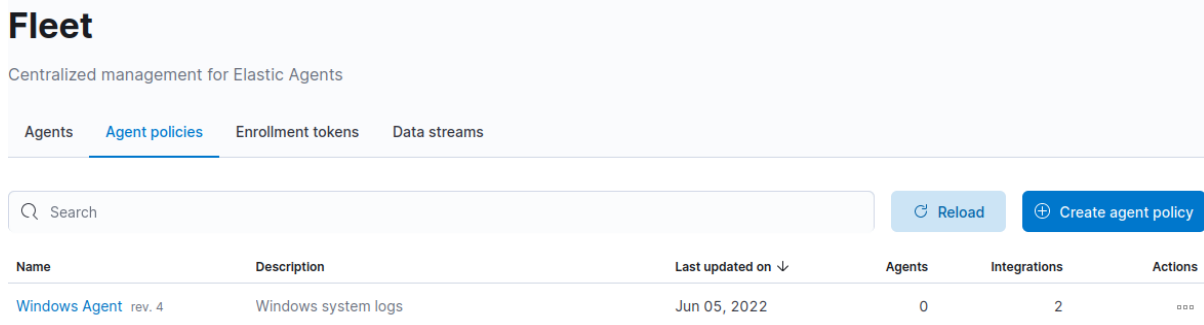


Figure 54 - Elastic agent policies

For this project, a single policy has been created to collect data from a Windows operating system. Specifically, as can be seen in figure 54, two different integrations have been added:

- **Endpoint Security:** provides prevention against malware threats and malicious actors. The information collected by this integration includes process executions, domain lookups, TCP connection details, file access or even network communication.
- **Windows:** collect logs and metrics from Windows OS and services.

Windows Agent Revision **4** | Integrations **2** | Used by **0 agents** | Last updated on **Jun 05, 2022** | [Actions](#) ▾

Windows system logs

[Integrations](#) [Settings](#)

Search... Namespace ▾ [Add integration](#)

Name ↑	Description	Integration	Namespace	Actions
endpoint-security		Endpoint Security v1.0.0	default	...
windows-1		Windows v1.5.0	default	...

Figure 55 - Windows agent integrations

From this moment on, we can deploy the Elastic Windows agent on the virtual machines. To do so, when a user creates and start a new guest (7.6.4 Start VMs), Tartarus server will automatically provision and install this agent through Vagrant[19].

Once the agent has been installed, it will communicate and synchronize with the Fleet server to receive the policy and integrations, configure itself and start sending logs and metrics to Elasticsearch.

Fleet

Centralized management for Elastic Agents

[Agents](#) [Agent policies](#) [Enrollment tokens](#) [Data streams](#)

Search Status ▾ | Agent policy **3** ▾ | Upgrade available [Add agent](#)

Showing 2 agents ● Healthy **2** ● Unhealthy **0** ● Updating **0** ● Offline **0**

Host	Status	Agent policy	Version	Last activity	Actions
PC-Marc	Healthy	Windows Agent rev. 8	7.14.1	13 seconds ago	...
fleet-server	Healthy	Default Fleet Server policy rev. 3	7.14.1	14 seconds ago	...

Figure 56 - Elastic agents

10.3 Kibana

In order to allow the user to observe the detection and behavior of a host against a threat, we will use the data collected by the agents and display it through Kibana interface.

The combinations and possibilities that Kibana offer us are endless. For that reason, in this section we will name the two most outstanding ones for this project.

10.3.1 Discover

In the *Discover* page we can view the events obtained from the agents. Additionally, you can quickly search and filter the data to find the information you are looking for in a simpler way.

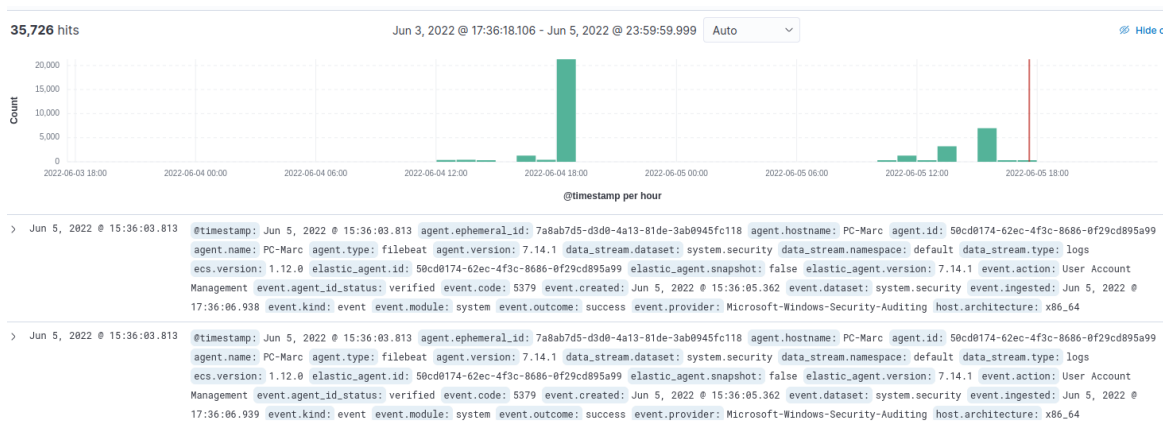


Figure 57 - Event display page

10.3.2 Alerts

In the *Alerts* page we can define or use prebuilt rules that throw an alert if some conditions are detected. These conditions are based on queries made on the data collected by the agents that is stored in elasticsearch, such as: CPU utilization, launched processes or network communications.

Alerts

Last alert: 23 hours ago

Manage detection rules



Figure 58 - Alert display page

In addition, we can obtain detailed information of an alert. For example, the process tree that has generated it (Figure 59).

All Process Events	
Process Name	Timestamp
winit.exe	Jun 4, 2022 @ 16:11:39.584
services.exe	Jun 4, 2022 @ 16:11:39.753
svchost.exe	Jun 4, 2022 @ 16:11:39.832
dllhost.exe	Jun 4, 2022 @ 18:26:12.789
0545f842c...	Jun 4, 2022 @ 18:26:13.250
ANALYZED EVE... cmd.exe	Jun 4, 2022 @ 18:26:23.511
conhost.exe	Jun 4, 2022 @ 18:26:23.623

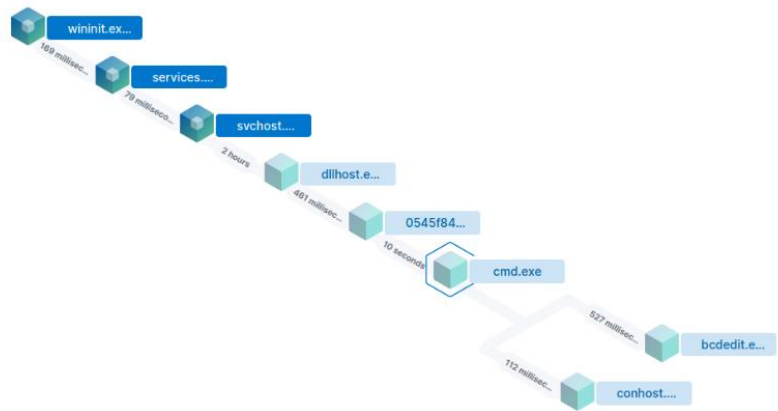


Figure 59 - Alert process tree

11 | Analysis of the results

In this section we will present several threat simulations examples to observe the complete operation of Tartarus and the interaction between all the modules that have been explained during this report.

11.1 LockBit 2.0

11.1.1 Introduction

LockBit is a ransomware group that follows a Ransomware-as-a-Service business model. That means that the developers of this malware customize its ransomware and lets other groups use the tool to encrypt and attack companies as they wish. This ransomware first appeared in September 2019 and in June 2021, the group rebranded to LockBit 2.0.

Focusing on the characteristics of this malware, LockBit is mostly spreading itself with Remote Desktop Protocol, Phishing Emails and Drive-by Downloads. It uses a hybrid-cryptography scheme of Elliptic Curve (*Curve25519*) and AES (*AES-128-CBC*) to encrypt the files of the victim machine. In addition, the rebranded version of LockBit includes several new features, such as: self-propagation, removal of shadow copies, bypass User Account Control (UAC), ESXi support, and the printing of ransom notes via printers detected on the victim's network[20].

Finally, note that the group also prides itself on having the fastest encryption on the ransomware market[21]. This is because it uses a multithreaded approach in the encryption phase and only 4 KB of data is encrypted per file.

11.1.2 Simulation

The steps required to configure and execute the Lockbit simulation are:

First of all, we need to define a new profile for the I4Tables tool. This profile includes an *iplist* file with the IPs to test if the system is properly isolated and a *rules* configuration file with the list of domains and subdomains that we want to allow communication.

In this case, since the ransomware doesn't have to communicate with any domain during its execution, we only need to add a static rule to access to the Tartarus server.

```
rbarcelo@rbarcelo-PC:~$ cat iplist.txt
8.8.8.8
10.0.0.1
192.168.1.1
1.1.1.1
```

Figure 61 - iplist file for Lockbit simulation

```
rbarcelo@rbarcelo-PC:~$ cat rules.txt
# File to configure i4tables resolutions
incide-server.local;192.168.101.1
```

Figure 60 - rules file for Lockbit simulation

Once we have created the files, we can import it to Tartarus using the CLI using the following command:

```
tartarus >> infrastructure network profiles create --name Lockbit --rules /home/rbarcelo/rules.txt --iplist /home/rbarcelo/iplist.txt Profile has been created.
```

Figure 62 – Creation of LockBit profile

The next step is to start the network virtual machine and the I4Tables tool with the profile created before.

```
tartarus >> infrastructure network vm start I4tables VM started.
```

Figure 63 - Network VM startup

```
tartarus >> infrastructure network i4tables start --profile Lockbit I4tables started.
```

Figure 64 - I4Tables tool startup

Then, we need to start the virtual machine where we are going to execute the malware sample. During this process, two different agents are installed in the machine: on one hand, the Elastic agent to collect logs and metrics of the system. On the other hand, the Tartarus agent that will automatically start asking the server if it has any pending task.

```
tartarus >> infrastructure vm start --name PCSIM-3 VM has been started.
```

Figure 65 - VM startup

At this point we already have all the infrastructure configured and we are ready to deploy the threat simulation.

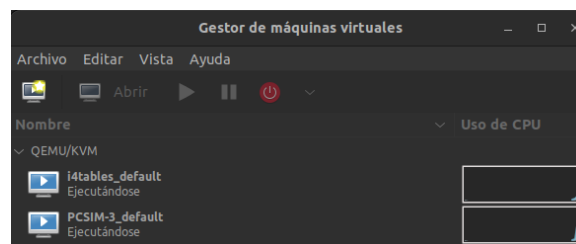


Figure 66 - Overview of the VMs

Therefore, we can start configuring the ransomware attack. To do this, we must prepare a YAML file with the characteristics of the threat, including the name of the machine where we want to execute the malware and the Lockbit sample that we have obtained in the online malware sandbox website called *any.run*[22].

```

1 ---
2 Task1:
3   name: "Task-Lockbit"
4   type_task: "execution"
5   vms:
6     - "PCSIM-3"
7   payloads:
8     - "0545f842ca2eb77bcac0fd17d6d0a8c607d7dbc8669709f3096e5c1828e1c049.exe"
9   commands:
10    kill: "no"
11
12

```

Figure 67 - LockBit task specifications

Finally, we have to create the task in Tartarus.

```

tartarus >> simulation tasks import --yaml /home/rbarcelo/lockbit.yaml --path /home/rbarcelo/lockbit/
Task has been imported and scheduled.

```

Figure 68 - LockBit task imported into Tartarus

At this point, the task is already scheduled in Tartarus waiting for the *PCSIM-3* agent to contact the server and request a new task.

Once the agent receives all the information, it starts running the malware and all the files stored the computer are encrypted. It should be noted that since the virtual machine is connected to an isolated network protected by I4Tables, it is a safe simulation and other hosts in the network are not in danger.

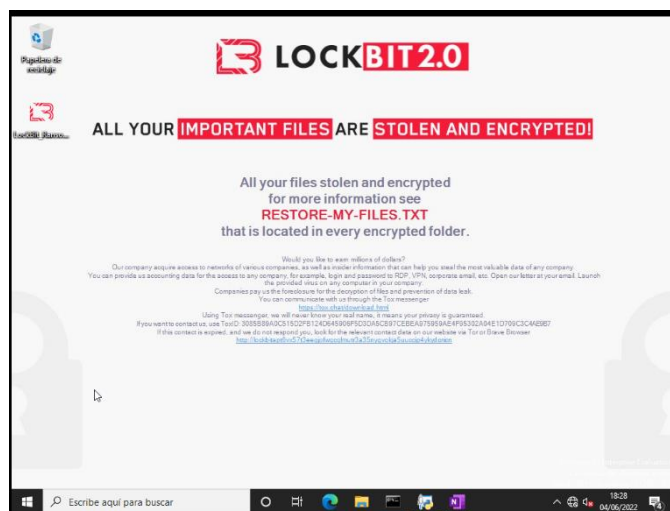


Figure 69 – Wallpaper modified by LockBit

11.1.3 Results

Once the ransomware has been executed, the user must observe the functioning of the malware and check whether the system has been able to block it or not.

To accomplish this, we will access the Kibana dashboard and review the alerts and logs generated.

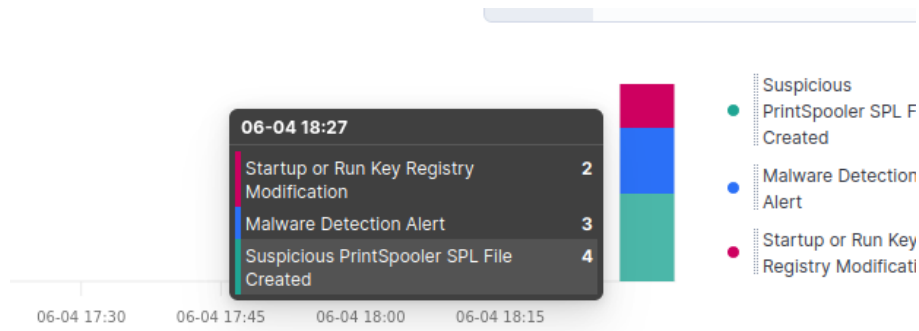


Figure 70 – Alerts generated by LockBit

As can be seen in the Figure 70, different alerts have been generated during the execution of Lockbit ransomware. Focusing on the alert *Malware Detection*, we have the possibility to observe the process flow, where we can confirm that this computer was unable to block the malicious sample. It should be noted that this computer does not have any antivirus or EDR installed to protect it.

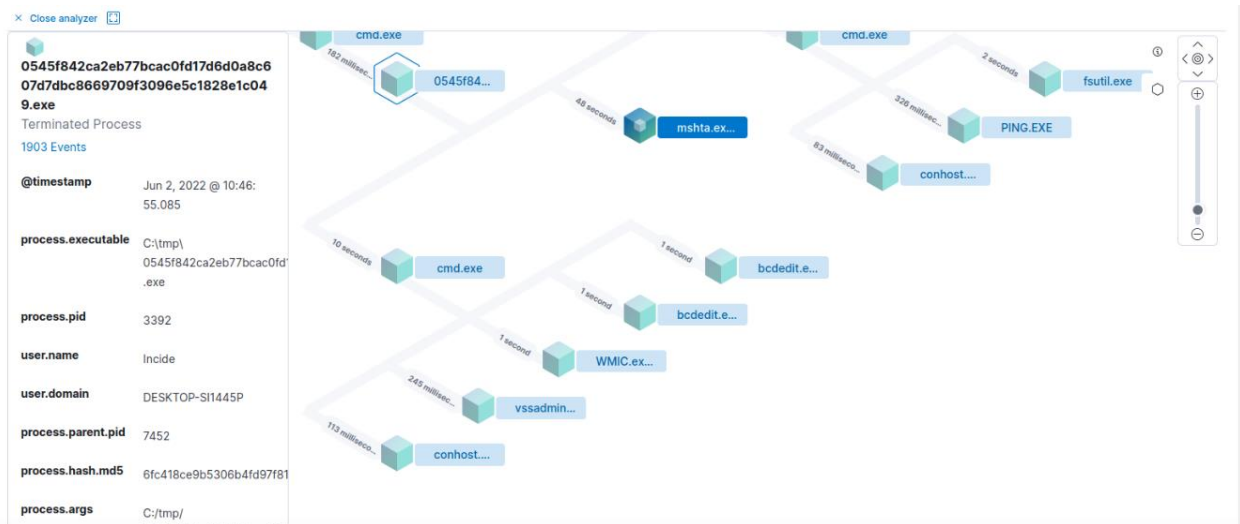


Figure 71 - LockBit process flow

Furthermore, we can review the events that have been registered during the execution of the threat. In Figure 72 we can observe that there are a large number of events related to the files, since it is a ransomware simulation.

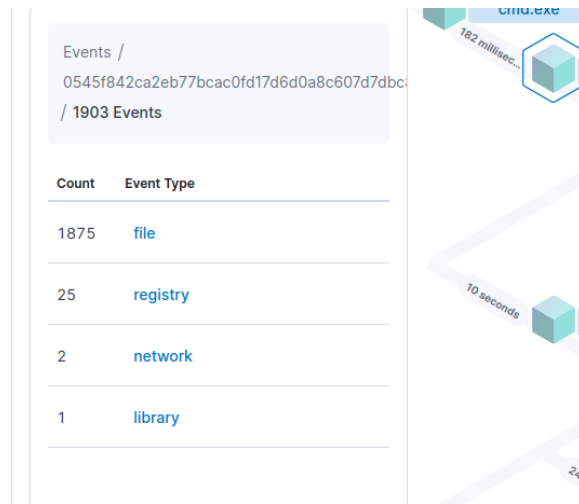


Figure 72 – General LockBit events

If we check the details of the events related to the files, we can confirm that it is the Lockbit ransomware because it is making changes to the files of the virtual machine, adding the ".lockbit" extension to them.

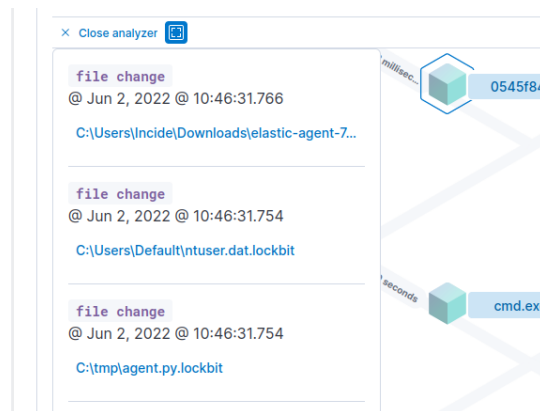


Figure 73 - LockBit file events

11.2 Follina

11.2.1 Introduction

In late May 2022, an independent cybersecurity research group called Nao_Se discovered a serious vulnerability in Microsoft products that allow attackers to execute remote arbitrary code (RCE). MITER has named this vulnerability CVE-2022-30190[23]; however, researchers called it Follina.

This is a Zero-Day vulnerability that has been detected in the Microsoft Windows Support Diagnostic Tool (MSDT), which is an application included in Windows that collects diagnostic information to send to Microsoft when an operating system crash occurs.

At first this does not seem like a big problem, unfortunately, due to the implementation of this tool, the vulnerability can be easily exploited via a malicious crafted Office document.

This infected document use the Word remote template feature to retrieve an HTML file from a remote web server, which in turn contains a JavaScript code that uses the ms-msdt protocol to load and execute PowerShell code within Windows[24]. Moreover, it should be noted that it can act even if the macros are disabled.

Consequently, an attacker who successfully exploits this vulnerability can run arbitrary code with the privileges of the calling application. Therefore, the actions that can be done in the post-exploitation phase are infinite: install programs, view, change, or delete data, or even create new accounts in the context allowed by the user's rights.

11.2.2 *Simulation*

In this section, we will explain the steps to configure and run the Follina threat simulation in Tartarus. The process is very similar to the one discussed previously, therefore we will not explain all the steps in detail.

It should be noted that the detonated sample is a Proof of Concept (PoC). Therefore, it does not perform any malicious action, it just writes a file in the system. However, it allows the analyst to check whether the system is vulnerable or not to the threat.

Regarding the steps to configure the simulation, we start by creating a new profile for the I4Tables tool. In this case, the malicious payload only needs to communicate with Tartarus and a webserver where the malicious HTML is hosted.

```
rbarcelo@rbarcelo-PC:~$ cat iplist.txt
8.8.8.8
10.0.0.1
192.168.1.1
1.1.1.1
```

Figure 74 - iplist file for Follina simulation

```
rbarcelo@rbarcelo-PC:~$ cat rules
#File to configure i4tables resolutions
incide-server.local;192.168.101.1
```

Figure 75 - rules file for Follina simulation

Then, we need to start the network virtual machine, the I4Tables tool and the virtual machine where the simulation will take place.

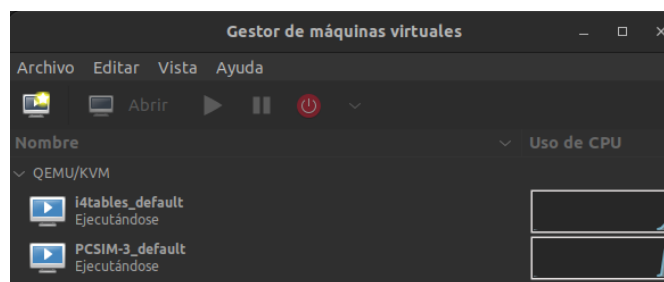


Figure 76 - Overview of the VMs

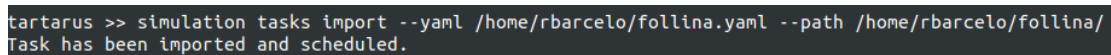
Now we can prepare the YAML file with the characteristics of the simulation, including the name of the machine where we will execute the threat and the infected word document that we have crafted to exploit the Follina vulnerability.



```
1 ---
2 Task1:
3   name: "Task-Follina"
4   type_task: "execution"
5   vms:
6     - "PCSIM-3"
7   payloads:
8     - "clickme.docx"
9   commands:
10    kill: "no"
11
12
```

Figure 77 - Follina task specification

Finally, we can create the task in Tartarus server, executing the following command:



```
tartarus >> simulation tasks import --yaml /home/rbarcelo/follina.yaml --path /home/rbarcelo/follina/
Task has been imported and scheduled.
```

Figure 78 - Follina task imported into Tartarus

At this point, the task is already scheduled, waiting for the agent of the virtual machine *PCSIM-3* to contact to the server and request a new task.

When the agent receives the task, it will download and open the malicious word document. As can be observed in the following picture, it is communicating with a remote server to retrieve an HTML file.

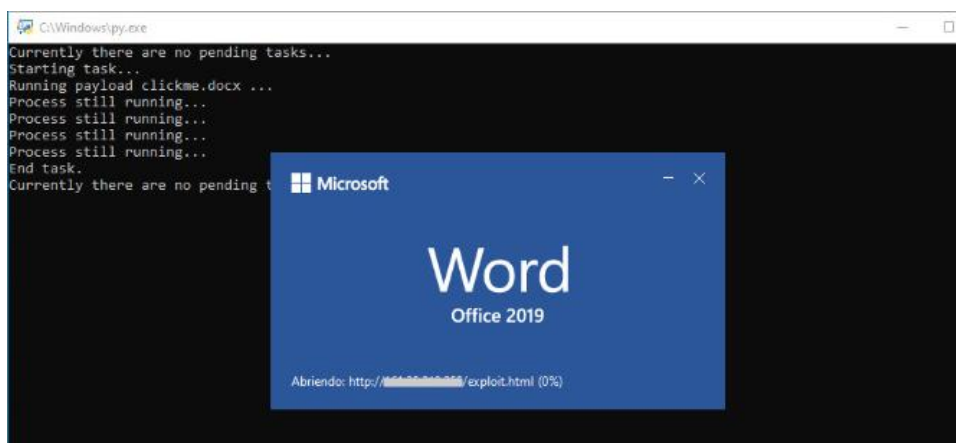


Figure 79 - Word document retrieving an HTML file from a webserver

Then, a troubleshoot window appears in the virtual machine, which internally executes a shell command.

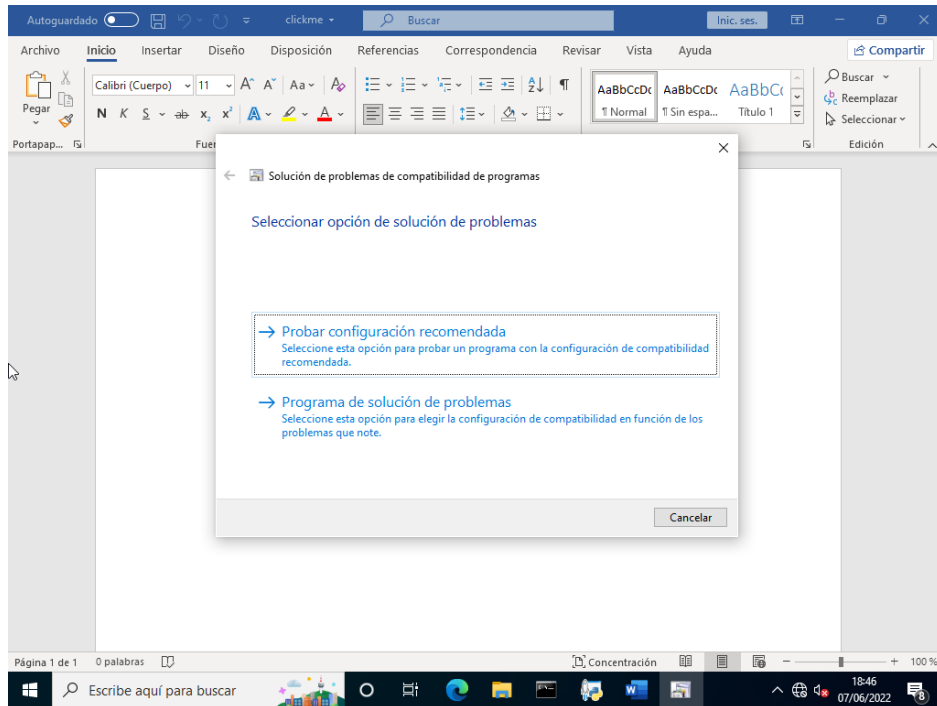


Figure 80 - Microsoft Windows Support Diagnostic Tool opened by a Word document

Finally, we can observe that this computer is vulnerable because the word document has been able to create a TXT file through the Microsoft Windows Support Diagnostic Tool.

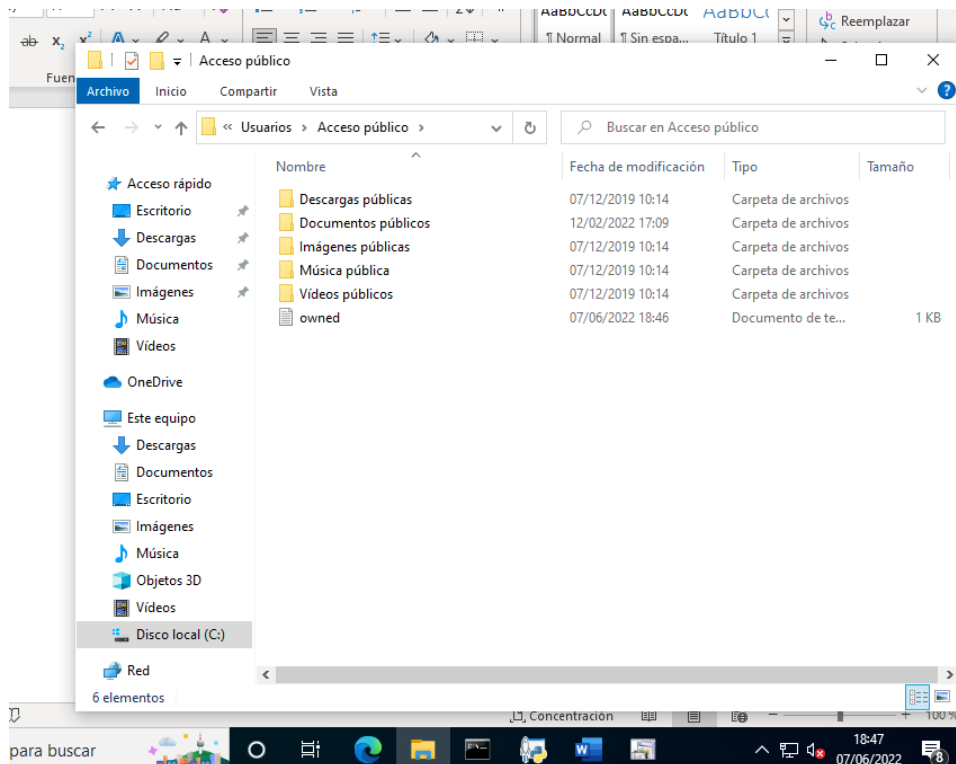


Figure 81 - File created by Follina exploit

11.2.3 Results

Once the threat simulation has finished, the user should observe the functioning of the malware and check whether the system has been able to block it or not. To accomplish this, we will access to Kibana web interface and review the alerts and logs generated.

First, we can find that an alert (*Execution from Unusual Directory – Command Line*) has been generated while the crafted Word document was being opened. This alert identifies process execution from suspicious default Windows directories.

@timestamp ↓ 1	Rule	Versl...	Method	Severity	Risk Score	event.module	event.action	event.category	host.name
Jun 7, 2022 @ 18:49:44.236	Execution from Unusual Di...	4	eq1	medium	47	endpoint	start	process	PC-Marc

Figure 82 – Alert generated by Follina

If we analyze the process tree of the mentioned alert, we can see that the Microsoft Windows Support Diagnostic Tool has been launched and it has interacted with a shell (*cmd.exe*). Therefore, here we can already observe suspicious behavior.

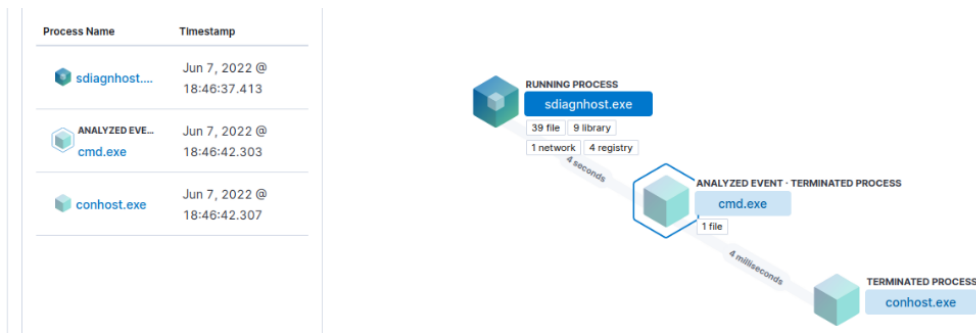


Figure 83 - Follina process flow

If we look at the details of the *cmd.exe* process, we can see that it has created a TXT file in the *Public* user’s home folder.

process.executable	C:\Windows\System32\cmd.exe
process.pid	724
user.name	Marc
user.domain	PC-MARC
process.parent.pid	2528
process.hash.md5	8a2122e8162dbef04694b9
process.args	C:\windows\system32\cmd.exe
process.args	/c
process.args	echo
process.args	owned
process.args	>
process.args	c:\users\public\owned.txt

Figure 84 - Details of cmd.exe process

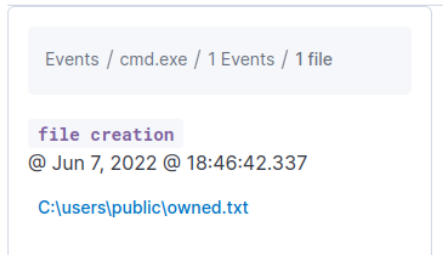


Figure 85 - File events of cmd.exe process

Furthermore, if we access to the *Discover* page in Kibana we can review the logs collected by the Elastic Agent from the virtual machine during the execution of the threat. Here we can also find relevant information.

For example, we can find logs that specified a connection to an IP, which is the IP of the server where the HTML file is hosted.

```
Jun 7, 2022 @ 18:46:37.454 destination.address: [redacted] @timestamp: Jun 7, 2022 @ 18:46:37.454 agent.id: 2ecab191-3962-4df2-b436-9e5ba132c12e agent.type: endpoint agent.version: 7.14.1
data_stream.dataset: endpoint.events.network data_stream.namespace: default data_stream.type: logs destination.geo.city_name: Frankfurt am Main
destination.geo.continent_name: Europe destination.geo.country_iso_code: DE destination.geo.country_name: Germany destination.geo.location: { "coordinates": [ 8.6843,
50.1188 ], "type": "Point" } destination.geo.region_iso_code: DE-HE destination.geo.region_name: Hesse destination.ip: 161.35.218.255 destination.port: 80
ecs.version: 1.6.0 elastic.agent.id: 2ecab191-3962-4df2-b436-9e5ba132c12e event.action: connection_attempted event.agent_id_status: verified event.category: network
```

Figure 86 - Network log generated during Follina execution

If we display the details of this log, we can see that this connection has been made from a Word document. Which is unusual and suspicious behavior.

host.os.platform	windows
host.os.version	2009 (10.0.19044.1645)
message	Endpoint network event
network.direction	outgoing
network.transport	tcp
network.type	ipv4
process.entity_id	MmVjVWIxOTEtMzk2Mi00ZGYyLWI0MzYtOWU1YmExMzJjMTJlLTQzNjQtMTMyOT
process.executable	C:\Program Files\Microsoft Office\root\Office16\WINWORD.EXE

Figure 87 - Details of network log

Therefore, with all this information, it is possible to understand the operation of the malware and confirm that the system is vulnerable to the Follina vulnerability.

12 | Conclusion

To conclude this project, it must be verified whether the objectives set at the beginning of the development have been achieved, as well as detailing the conclusions drawn from the results obtained.

The main objective of this project was to design and develop an application which can distribute and execute malware samples into sandboxed virtual machines, retrieve metrics of the results of each execution and allow to observe the detection and blocking capacity of the machine.

This objective has been successfully achieved thanks to the development of Tartarus application that has been detailed throughout this thesis.

To conclude, Incide has been able to significantly improve and automate the Imminent Threat Exposure service thanks to this tool. Once the infrastructure has been configured in Tartarus, preparing and running the threat simulations can be done in just a few steps. In addition, as each agent is independent from the others, we can parallelize the execution of the malware samples on several virtual machines at the same time, reducing the necessary time that the analyst must invest on this task. Finally, since the interaction with the Tartarus server is done through commands, we can create scripts that further automate the execution of the threat simulations.

12.1 Future development

This section details some of the improvements that can be implemented to Tartarus in the future, with the aim of improving the application and adding new functionalities. It should be noted that due to the modular architecture, the improvements are practically endless.

12.1.1 Secure communication

Currently the requests made by the Command-Line and the Tartarus agent to the server are not protected, since they are made through HTTP. That means that people could eavesdrop the data that is being transferred.

Therefore, encrypted connections should be implemented to add an extra layer of security. To achieve this, we can add SSL certificates so we can communicate with the server endpoints via HTTPS.

12.1.2 Authentication

The second functionality that can be added in the future is the authentication of users. Currently there is no process to register or log in into the application. Therefore, any user who has access to the server can use Tartarus application and perform any type of action.

For this reason, an authentication process using JWT Tokens along with a method to manage user roles will be incorporated, to create different type of users and restrict access to certain actions.

12.1.3 Generate reports

Currently the process of checking if the malware sample has been executed or blocked must be done using the Elastic tools. Then, if the user needs to create a report with the results obtained of the simulation to send it to the client, he must write it manually with all the collected information.

In the future, this process could be improved by creating a function that automatically generates the reports that the user requests. One way to solve this is to create a document template that is automatically filled in by Tartarus with the information obtained from Elastic.

13 | Bibliography

- [1] "2021 Cyber Attack Trends," *Check Point Software*.
<https://pages.checkpoint.com/cyber-attack-2021-trends.html> (accessed Jun. 06, 2022).
- [2] "Incide | Ciberseguridad-DFIR-Auditoría-Red Team." <https://www.incide.es/> (accessed May 22, 2022).
- [3] "Boxes | Vagrant by HashiCorp." <https://www.vagrantup.com/docs/boxes> (accessed Jun. 24, 2022).
- [4] "Python Click." <https://palletsprojects.com/p/click/> (accessed May 25, 2022).
- [5] "requests · PyPI." <https://pypi.org/project/requests/#description> (accessed May 26, 2022).
- [6] M. Martínez Gómez, "Implementación de un centro de operaciones de seguridad (SOC) de código abierto con elementos de red para sistemas industriales," Oct. 2021, Accessed: Jun. 03, 2022. [Online]. Available:
<https://riunet.upv.es:443/handle/10251/174344>.
- [7] "Elastic." <https://www.elastic.co/es/> (accessed May 25, 2022).
- [8] "Elasticsearch: El motor de búsqueda y analítica distribuido oficial | Elastic." <https://www.elastic.co/es/elasticsearch/> (accessed Jun. 03, 2022).
- [9] "Kibana: Explora, visualiza y descubre datos | Elastic." <https://www.elastic.co/es/kibana/> (accessed Jun. 03, 2022).
- [10] "Elastic Agent | Elastic." <https://www.elastic.co/es/elastic-agent> (accessed Jun. 03, 2022).
- [11] "What is Lean?" <https://www.lean.org/whatslean/> (accessed Sep. 27, 2020).
- [12] "Salario de Ingenieros informáticos en España." <https://es.indeed.com/career/ingeniero-informático/salaries> (accessed May 13, 2022).
- [13] "Salario de Jefe de proyecto en España." <https://es.indeed.com/career/jefe-de-proyecto/salaries> (accessed May 13, 2022).
- [14] "ONYX: ordenador portátil potente de alto rendimiento para profesionales." <https://www.mountain.es/portatiles/onyx> (accessed May 14, 2022).
- [15] "VSCodium - Open Source Binaries of VSCode." <https://vscodium.com/> (accessed May 14, 2022).
- [16] "Disk2vhd | Microsoft Docs." <https://docs.microsoft.com/en-us/sysinternals/downloads/disk2vhd> (accessed Apr. 17, 2022).
- [17] "vagrant-libvirt/create_box.sh." https://github.com/vagrant-libvirt/vagrant-libvirt/blob/master/tools/create_box.sh (accessed Apr. 18, 2022).
- [18] "Integraciones de Elasticsearch | Elastic." <https://www.elastic.co/es/integrations/> (accessed Jun. 05, 2022).
- [19] "vagrant provision - Command-Line Interface | Vagrant by HashiCorp." <https://www.vagrantup.com/docs/cli/provision> (accessed Jun. 09, 2022).

- [20] C. Dong, "LockBit Ransomware v2.0." <https://chuongdong.com/reverse-engineering/2022/03/19/LockbitRansomware/> (accessed Jun. 04, 2022).
- [21] "Malware Evolution - Analyzing LockBit 2.0." <https://www.cynet.com/attack-techniques-hands-on/malware-evolution-analyzing-lockbit-2-0/> (accessed Jun. 04, 2022).
- [22] "Lockbit sample | ANY.RUN." <https://any.run/report/0545f842ca2eb77bcac0fd17d6d0a8c607d7dbc8669709f3096e5c1828e1c049/2018f8e3-1571-4b4a-89fb-f19228736962> (accessed Jun. 04, 2022).
- [23] "CVE-2022-30190 - Security Update Guide - Microsoft - Microsoft Windows Support Diagnostic Tool (MSDT) Remote Code Execution Vulnerability." <https://msrc.microsoft.com/update-guide/en-US/vulnerability/CVE-2022-30190> (accessed Jun. 10, 2022).
- [24] "Analizando y explotando FOLLINA (CVE-2022-30190)." <https://ciberseguridad.blog/analizando-y-explotando-follina-msdt-cve-2022-30190/> (accessed Jun. 10, 2022).