Computer Engineering Department

C2 Framework “C2nan” Deployed on a Raspberry Pi

Graduation project is submitted to Computer Engineering Department in Partial fulfilment of the requirements for the Degree of Bachelor of Computer Engineering

BY

Kareem Mostafa Alsadeq

Abdelaziz Alaaldin Ali Abdelaziz Elhawary

Tamer Khaled Fathy Ibrahim

Mohamed Mahmoud Abdellah

Salma Mohamed Salah Abdelatty Elkomy

Rana Ahmed Mahmoud

SUPERVISED BY

Assoc. Prof. Samir Gaber Abdelgawad | Dr. Amany Mohamed Saleh

Assoc. Prof. Amr Mohamed Elsayed

Acknowledgment

We would like to express our deepest gratitude and appreciation to all those who have contributed to the completion of this documentation. Their support, guidance, and encouragement have been invaluable throughout the entire process of our journey.

First and foremost, we are immensely grateful to our supervisors, **Assoc.Prof.Samir Gaber Abdelgawad, Assoc.Prof.Amr Mohamed Elsayed and Dr.Amany Mohamed Saleh**, for their unwavering support, expertise, and valuable insights. Their guidance and mentorship have been instrumental in shaping this project and pushing it toward excellence.

We would also like to thank the mentor who helped us, **Eng. Ahmed Hesham**, for his valuable feedback, constructive criticism, and expertise. His insightful comments and suggestions have significantly enhanced the quality and rigor of this Project.

We are deeply indebted to the participants who generously contributed their time, insights, and expertise to this project. Their willingness to share their knowledge and experiences has been instrumental in enriching the content and validity of this documentation.

Lastly, we want to express our heartfelt gratitude to our families for their unwavering love, understanding, and encouragement. Their constant support has been a source of strength and motivation throughout this endeavor. This Project would not have been possible without the collective efforts, guidance, and support of these individuals and organizations. We are truly grateful for their contributions, and We humbly acknowledge their invaluable role in making this project a reality.

Abstract

Command and Control (C2) frameworks play a crucial role in bolstering cyber defense strategies, enabling efficient monitoring, detection, and mitigation of cyber threats. This document presents a case study showcasing a prototype C2 framework named "C2nan," providing a comprehensive understanding of its architecture, functionalities, and potential application scenarios. C2nan serves as a practical tool for red-teaming exercises, highlighting key components such as command servers, agents, and tasks. The document explores the client-server architecture and elucidates how security engineers leverage C2 frameworks to manage compromised machines, identify vulnerabilities, and fortify system defenses. Moreover, it outlines the diverse functionalities offered by C2 frameworks, including command execution, data collection, and task scheduling, and reporting. Developed in Python-Django, C2nan audits Windows and Linux operating systems, along with Active Directory environments, employing red-teaming methodologies, automation capabilities, and third-party integrations. This framework serves as a valuable example, providing insights into the workings of C2 frameworks and facilitating a comprehensive understanding of their operational dynamics.

List of Abbreviations

AD - Active Directory

AMSI - Anti-Malware Scan Interface

API - Application Programming Interface

APT - Advanced Persistent Threat

ARM - Advanced Risk Machine

CA - Certificate Authorities

C2 - Command and Control

CSS - Cascading Style Sheets

CT - Certificate Transparency

DAI - Dynamic ARP Inspection

DB - Data Base

DC - Direct Current

DHCP - Dynamic Host Configuration Protocol

DLL - Dynamic Link Library

DNS - Domain Name System

DV - Domain Validation

EV - Extended Validation

GB - Giga Byte

GPP - Group Policy Preferences

GUI - Graphical User Interface

HDMI - High-Definition Multimedia Interface

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

IDS - Intrusion Detection System

IP - Internet Protocol

IOT - Internet of Things

IPS - Intrusion Prevention System

JSON - JavaScript Object Notation

MAC - Media Access Control

MITM - Man-in-the-Middle

MVT - Model Template View

NASA - National Aeronautics and Space Administration

NBC - National Broadcasting Company

NFL - National Football League

NHL - National Hockey League

NSA - National Security Agency

ORM - Object-Relational Mapper

OS - Operating System

OV - Organization Validation

Pen Tester - Penetration Tester

PNAC - Port-Based Network Access Control

RAT - Remote Access Terminal

RAM - Random-Access Memory

RPI - Raspberry Pi

SAS - Secure Attention Sequence

SFTP - Secure File Transfer Protocol

SSH - Secure Shell

SSHFS - Secure Shell File System

SSL - Secure Sockets Layer

TCP/IP - Transmission Control Protocol/Internet Protocol

URL - Uniform Resource Locator

USB - Universal Serial Bus

VNC - Virtual Network Computing

VPN - Virtual Private Network

WebDAV - Web-based Distributed Authoring and Versioning

WHO - World Health Organization

List of figures

Table of Contents

Chapter 1
Introduction

Chapter 1: Introduction

1.1 Background

As technology becomes increasingly integrated into our daily lives, the demand for cybersecurity engineers has reached a critical level. In 2021, The **FBI's Internet Crime Complaint Center (IC3)** released the Internet Crime Report, which indicates that the number of victims of Internet crime and the amount of money lost due to such activities reached a new peak. A staggering 847,376 complaints were logged by IC3 in the last calendar year, resulting in total losses amounting to **$6.9 billion**. Phishing, vishing, smishing, and pharming were among the most frequently recorded internet crimes in 2021. [1]

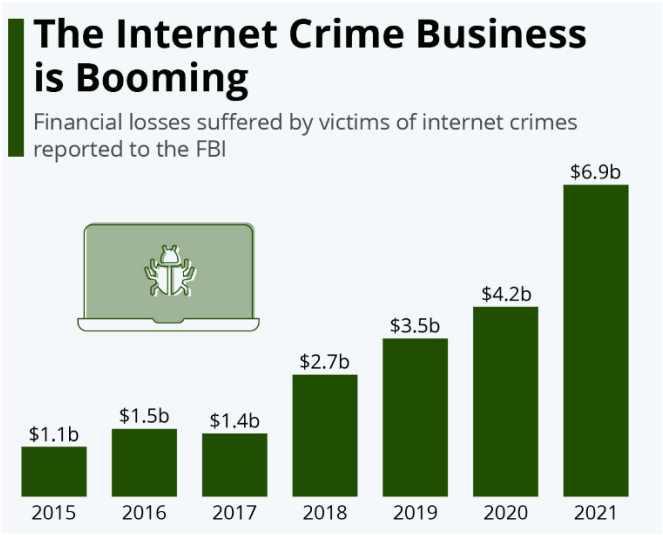

Figure 1 cyber crime Statistics

Also, the **Center for Strategic & International Studies (CSIS)** tracks "cyber-attacks on government agencies, defense, and high-tech companies, or economic crimes with losses of more than a million dollars." Over the past decade, they've tracked 490 significant cyber incidents. [2]

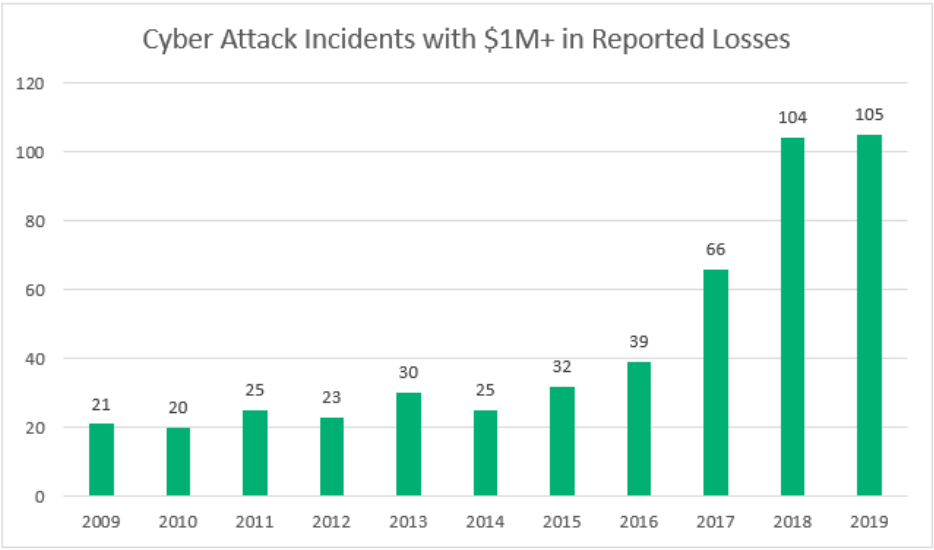

Figure 2 cyber-attacks losses

Moreover, the rise of emerging technologies like the Internet of Things (IoT), artificial intelligence, and cloud computing has introduced new vulnerabilities that necessitate specialized knowledge and skills for effective protection. The need to secure sensitive information and critical infrastructure has elevated the importance of cybersecurity engineers, and this demand is anticipated to continue growing in the coming years.

These cybersecurity professionals play a vital role in safeguarding computer systems, networks, and data from unauthorized access, theft, and damage. By implementing security measures like firewalls, encryption, and access controls, cyber security engineers ensure the protection of systems and information from a wide range of cyber threats, such as ransomware, phishing attacks, and data breaches.

To address security concerns comprehensively, organizations often establish two teams: the **blue team** and the **red team**.

1.2 Blue Team

In the world of cybersecurity, the "blue team" refers to the defenders who work to protect computer systems, networks, and data from cyber threats. The blue team's scope of work is broad and encompasses a range of activities aimed at preventing, detecting, and responding to cyber-attacks. The primary focus of the blue team is to maintain the confidentiality, integrity, and availability of systems and data. One of the main responsibilities of the blue team is to develop and implement security policies, procedures, and controls to ensure the protection of systems and data from cyber threats. This includes conducting risk assessments to identify vulnerabilities and threats and subsequently implementing measures to mitigate those risks. The blue team also ensures that systems and networks are configured securely, with appropriate access controls, firewalls, and intrusion detection systems in place. Another key aspect of the blue team's work is monitoring and detecting potential security incidents. This involves the utilization of various tools and technologies to monitor network traffic, identify anomalies, and detect potential security breaches. Additionally, the blue team conducts security assessments and penetration testing to identify vulnerabilities and weaknesses in systems and networks. In the event of a security incident, the blue team is responsible for responding quickly and effectively to mitigate the impact of the attack.

Figure 3 Blue team logo

This includes conducting a forensic investigation to determine the cause of the incident and taking steps to prevent its recurrence in the future. The blue team plays a critical role in ensuring the security of computer systems, networks, and data. Through the implementation of proactive security measures, the detection and response to security incidents, and the continuous improvement of security practices, the blue team helps protect organizations from the escalating threat of cyber-attacks.

1.3 Red Team

Figure 4 Red team logo

The "red team" comprises cybersecurity professionals whose primary role is to simulate real-world cyber-attacks against organizations. The red team's objective is to test the effectiveness of an organization's security measures and identify vulnerabilities that may remain undetected through regular security testing. To achieve this, the red team utilizes various tools and techniques, including social engineering, penetration testing, and vulnerability scanning, to gain unauthorized access to the organization's systems and data. An essential goal of the red team is to pinpoint vulnerabilities that potential attackers could exploit. This involves identifying weaknesses in network security, such as misconfigured firewalls and open ports, as well as vulnerabilities present in software applications and operating systems. The red team also focuses on assessing human factors, such as identifying employees who may be susceptible to phishing attacks or social engineering tactics.

The red team's work is highly technical, often employing specialized tools and techniques to identify and exploit vulnerabilities. For instance, they may utilize exploit frameworks like Covenant, Metasploit, or Cobalt Strike to gain system access or leverage password-cracking tools to breach weak passwords. Additionally, they may employ reverse engineering techniques to analyze malware and understand its capabilities. Once vulnerabilities are identified, the red team collaborates closely with the organization's blue team to assist in remediation efforts. This can include providing recommendations to enhance security controls. The red team plays a critical role in identifying and mitigating cybersecurity risks. By simulating real-world attacks, they aid organizations in identifying vulnerabilities and enhancing their overall security posture, thereby

fortifying defenses against cyber threats. The primary focus of the red team's work lies in conducting penetration testing to evaluate an organization's security resilience. [3]

1.4 Penetration testing

The penetration testing phases **provide a systematic approach** to testing and ensure that the process is comprehensive, accurate, and thorough. By following a well-structured approach, the penetration tester can identify vulnerabilities and security gaps in the target system and help the organization address them before attackers can exploit them.

It consists of 5 main phases:

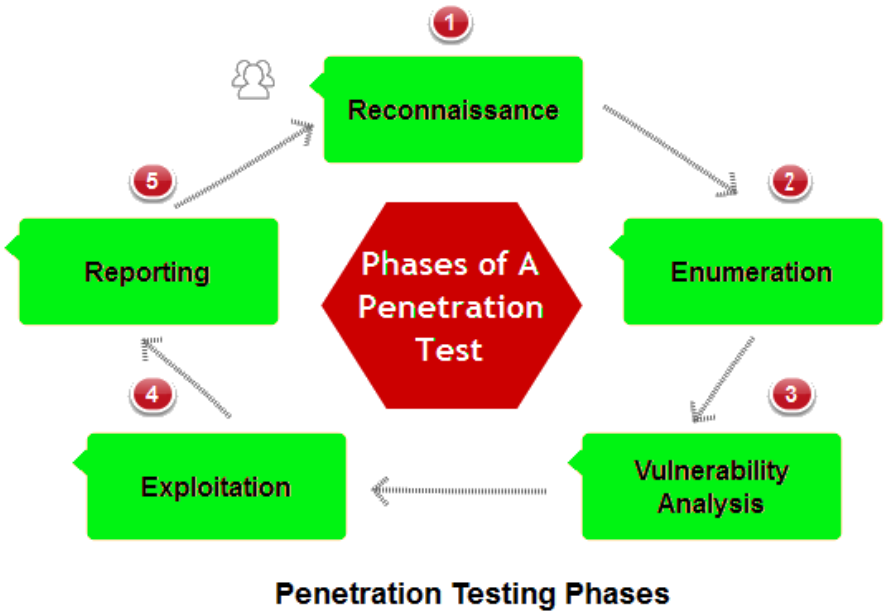

Figure 5 Penetration testing phases

1.4.1 Reconnaissance

Reconnaissance, also known as recon for short, is the process of gathering information about a target system or network. In the context of cyber security, reconnaissance is a critical phase of an attacker's attempt to gain unauthorized access to a network or system. Examples of reconnaissance techniques: Port Scanning, Social Engineering, Network Mapping, and Reconnaissance using Malware.

1.4.2 Enumeration

Once all the relevant data has been gathered in the reconnaissance phase, it's time to move on to enumeration. In this penetration testing phase, the tester uses various tools to **identify open ports and check network traffic on the target system.** Because open ports are potential entry points for attackers, penetration testers need to identify as many open ports as possible for the next penetration testing phase.

1.4.3 Vulnerability Analysis

The third penetration testing phase is vulnerability analysis, in which the tester uses all the data gathered in the reconnaissance and enumeration phases to **identify potential vulnerabilities and determine whether they can be exploited**. Much like enumeration, vulnerability analysis is a useful tool on its own but is more powerful when combined with the other penetration testing phases.

1.4.4 Exploitation

Once vulnerabilities have been identified, it's time for exploitation. In this penetration testing phase, the penetration tester attempts to **access the target system and exploit the identified vulnerabilities**, typically by using a tool like Metasploit to simulate real-world attacks.

1.4.5 Reporting

Once the exploitation phase is complete, the tester prepares **a report documenting the penetration test's findings.** The report generated in this final penetration testing phase can be used to fix any vulnerabilities found in the system and improve the organization's security posture.[4]

1.5 Red Team Assessment

A security engineer is given initial access to an internal device on the networkwhere the penetration testing process to the system begins. He then tries to apply red team assessment objectives.

The primary **objectives of red team assessments** include:

- Attaining the highest level of privilege within the system.

- Obtaining access to sensitive data wherever possible.
- Executing these operations on the maximum number of devices within the internal network.
- Documenting the discovered issues in a focused report.
- Minimizing the visibility of the stealth process. [5][6]

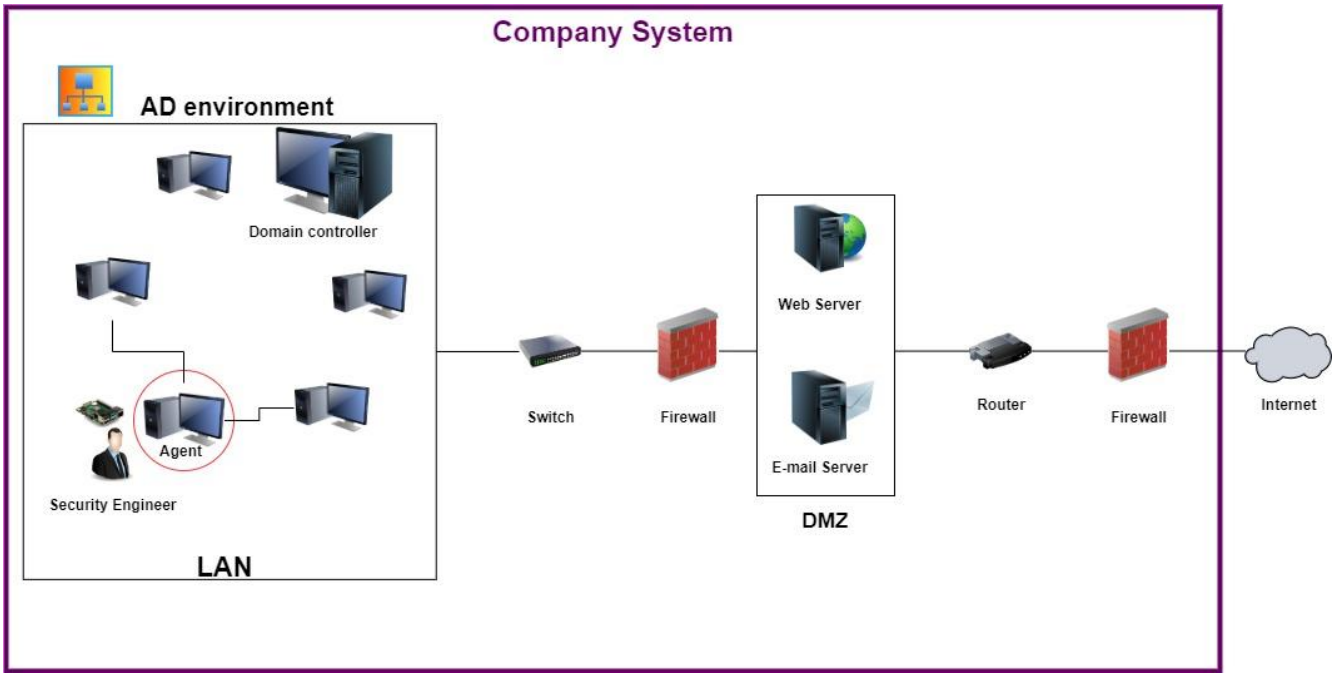

Figure 6 Company Environment

The ultimate aim of red team assessments is to test an organization's security controls and identify areas for improvement, thereby enhancing its overall security posture.

1.6 Problem Statements

In auditing large hierarchical organizations:

- Each operating system has its unique architecture, features, and vulnerabilities, which require a specific penetration testing strategy.
- For example, when testing Windows OS, a penetration tester may focus on testing Microsoft Active Directory, privilege escalation, and exploitation of buffer overflow vulnerabilities. On the other hand, when testing Linux OS, a tester may concentrate on securing the network services and configuring firewalls. Similarly.

- Therefore, it is crucial to develop a tailored penetration testing strategy for each OS to ensure that it is secure and resilient against cyber-attacks.
- Miss-auditing integrity may cause weakness in reports.
- It's required to audit more than one system or device. Each one has its techniques in penetration testing, and wasting time testing each device individually consumes time which presents the system to possible attacks that consume money.
- The results of a penetration test often contain a large amount of technical information. This information needs to be interpreted, prioritized and acted upon appropriately to improve the organization's security posture. Overall, handling the results of a penetration test requires a well-defined process, effective communication, and a clear understanding of the organization's security priorities and limitations.
- Writing a report after completing the penetration testing phases communicates the findings, identifies vulnerabilities, and provides recommendations for remediation. However, writing an effective and comprehensive report requires great expertise and attention to detail. One of the challenges of writing a report after a penetration test is the sheer volume of information gathered during the testing process.
- This information includes technical details such as the methodology used, vulnerabilities identified, and their severity levels. It also includes non-technical information such as the scope of the test, the objectives, and the findings.
- Another challenge is translating the technical information into terms that can be easily understood by stakeholders who may not have a technical background. The report should be clear, concise, and easy to follow.
- Additionally, the report should provide actionable recommendations for remediation that can be implemented by the organization.[7]

1.7 Project Objectives

A tool that:

- Familiarizing GUI for Red Teamer to use.
- Handles multiple agents at the same time without conflicts with results and tasks, whatever OS type.
- Provide multi-user collaboration so red teamers can use the tool at the same time.

- Providing distinct reports for each device.
- Integrity extensive third party.
- Advancing automation capabilities for each OS

1.8 Scope of Work

1.8.1 Command and Control (c2)

"Command and control" (C2) systems are used to manage remote sessions from compromised hosts. From a command and control program interface, a security tester can send commands directly from the program or access a remote shell. During a penetration test, a security tester can deploy a remote access terminal (RAT) on a compromised host that dials back to a command-and-control server." [8]

It just requires having an agent inside the system that communicates with the c2 server that provides persistency over large periods and almost all the capabilities an attacker would need to perform movement and other post-exploitation actions. There are a lot of free, open-source post-exploitation toolsets that provide this kind of capability, like Metasploit and others. [9]

1.8.2 About c2 servers/agents

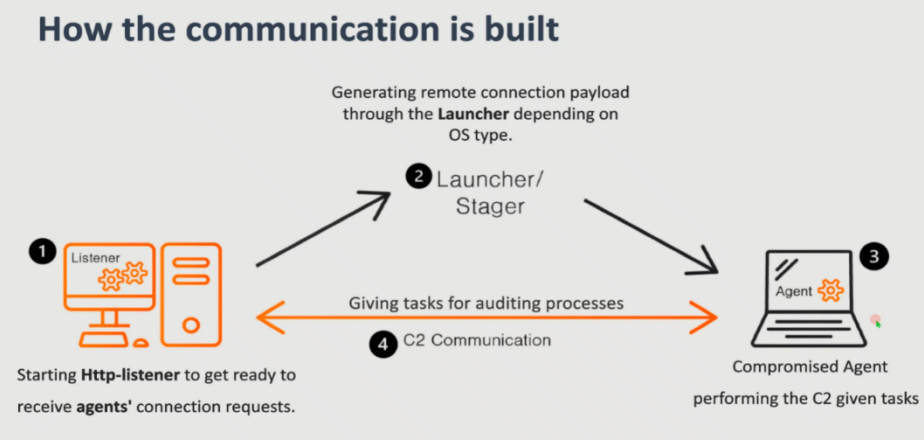

Figure 7 How C2 communicates.

Start HTTP listener

Starting an HTTP listener is the core of command and control (C2) operations in cybersecurity. An HTTP listener serves as a component of a C2 infrastructure, enabling red teamers to establish communication with compromised systems. The listener functions by opening a port on the compromised system, specifically configured to accept incoming HTTP requests. Once the listener is initiated, red teamers can transmit commands to the compromised system through HTTP requests, granting them the ability to execute arbitrary code, exfiltrate data, or carry out other malicious activities. The tool employed is configured to commence the listener and actively await incoming HTTP requests from the attacker's command server. In certain cases, the listener may also be equipped with encryption or other obfuscation techniques to evade detection by security tools.

In summary, the initiation of an HTTP listener is a crucial step in establishing a C2 channel within cybersecurity, frequently utilized in advanced persistent threat (APT) attacks and other sophisticated attack scenarios.

Launcher stage

A launcher stage is a vital component of a C2 infrastructure used to initiate and execute a payload on a compromised system. The payload typically encompasses malware, keyloggers, or other tools employed by the red teamer to assume control over the compromised system. The purpose of the launcher stage is to establish an initial foothold on the compromised system. Upon execution of the appropriate command specific to the operating system in question, the launcher stage commences, enabling the red teamer to leverage the C2 infrastructure for communication with the compromised system and execution of subsequent commands.

The compromised machine performs tasks.

Once communication is established, the compromised machine sends a signal to the red teamer's server, seeking its next instruction or task. The compromised machine then proceeds to execute the commands received from the attacker's C2 server, which may involve installing additional software. At this stage, the red teamer gains full control over the compromised machine, granting them the ability to execute any desired code. The malicious code typically propagates to other machines, thereby creating a botnet—a network of infected machines. Ultimately, the compromised machine sends the results back to the red teamer through the server. [10][11]

Overall:

A basic c2 server should be able to:

Start and stop listeners.

Generate payloads.

Handle agents and task them to do stuff.

An agent should be able to:

1. Download and execute its tasks.
2. Send results.
3. Persist.

A listener should be able to:

1. Handle multiple agents.
2. Host files.
3. And all communications should be encrypted.[12]

Chapter 2
Literature Review

Chapter 2: Literature Review

2.1 Related Work

2.1.1 0xRick's (Basic C2)

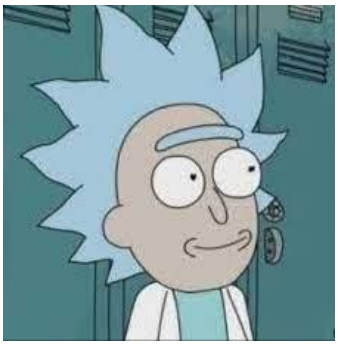

Figure 8 Basic c2 by 0Xrick

Overview:

- 0xRick is very basic as it solely establishes communication between the C2 server and the agent, without any tasks to send or results to receive.
- 0Xrick was built using Flask and Python.

In details:

LISTENER

Listeners serve as the backbone of the server by facilitating communication between the server and the agents. In the case of Basic C2, HTTP listeners and Flask are utilized to create the listener application. This entails establishing the requisite directories for storing files, encryption keys, and agents' data.

The Flask application, responsible for delivering the comprehensive functionality of the listener, consists of five routes: /reg, /tasks/<name>, /results/<name>, /download/<name>, and /sc/<name>.

The /reg endpoint handles new agents, accepting only POST requests. It requires two parameters: name (for the hostname) and type (for the agent's type). Upon receiving a new request, it generates a random string consisting of six uppercase letters as the new agent's name (which can be changed later). The endpoint retrieves the hostname and agent's type from the request parameters, as well as the remote address (the IP address of the compromised host). Using this information, it creates a new Agent object, saves it to the agent's database, and responds with the generated random name. This allows the agent on the other side to identify itself (although no tasks are sent initially, the agent can request tasks using this name).

The /tasks/<name> endpoint is used by agents to request and download their tasks. The <name> placeholder represents the agent's six uppercase letters name. Only GET requests are accepted.

The endpoint checks if there are new tasks available by verifying the existence of the tasks file. If new tasks are present, it responds with the tasks; otherwise, it sends an empty response.

The /results/<name> endpoint is used by agents to send their results. The <name> placeholder represents the agent's name. Only POST requests are accepted, with a single parameter: result, which contains the results.

The /download/<name> endpoint handles file downloads. The <name> placeholder represents the file name. Only GET requests are accepted. The endpoint reads the requested file from the designated file path and sends it to the requester.

The /sc/<name> endpoint serves as a wrapper around the /download/<name> endpoint specifically for PowerShell scripts. It responds with a download cradle that includes a one-liner to bypass AMSI. The one-liner downloads the original script from the /download/<name> endpoint, with <name> representing the script name. Only GET requests are accepted for this endpoint.

Start listeners in threads, and this causes problems as it can’t route between the port of the HTTP listener and different.

AGENT

When an agent is present on a system, the first step it takes is to retrieve the hostname of that system. It then sends a registration request to the server and awaits the assigned name. Following this, the agent enters an infinite loop, continuously requesting tasks and sending back results to the server.

Payload Generator

The payload generator also referred to as the launcher, generates a concise payload in the form of a one-liner. The generated payload can be either Bash for Linux or PowerShell for Windows. Throughout the implementation phase, we encountered a significant challenge with the Power Shell launcher. Specifically, when executing a command involving sub-processes, the launcher would wait for the sub-processes to finish and store the results in a buffer. Subsequently, the code was designed to extract and display the results from the buffer. However, this approach proved

problematic when the executed process was substantial, resulting in buffer overflow and channel crashes. [13]

2.1.2 Covenant

Covenant was constructed by a team of 12 cybersecurity professionals. It is a .NET command and control framework designed to showcase the attack surface of .NET, simplify the utilization of offensive .NET tradecraft, and function as a collaborative command and control platform for red teamers. The framework was developed using C# and specifically targets .NET Core, enabling Covenant to operate seamlessly across multiple platforms, including Linux, MacOS, and Windows. However, it is important to note that while Covenant is compatible with audit\windows OS, it may not support other operating systems. [14][15]

Figure 9 Covenant logo

Chapter 3
System Requirements & Specifications

Chapter 3: System Requirements and Specifications

3.1 Functional Requirements

3.1.1 Communication between pen-tester and agent:

Effective communication between a penetration tester (pen tester) and an agent within a command-and-control framework is crucial for conducting comprehensive security assessments and vulnerability testing. The pen tester, who operates the framework, relies heavily on clear and dependable communication channels with the agent deployed on target systems. This communication enables the pen tester to send commands, receive feedback, and obtain critical information from the agent throughout the assessment process.

Implementing secure and encrypted communication protocols ensures the confidentiality and integrity of the data exchanged, preventing unauthorized access or interception by malicious entities. By establishing robust communication, pen testers can engage in real-time interaction, execute targeted testing scenarios, and promptly respond to emerging findings. This seamless communication framework empowers pen testers to leverage their expertise and utilize the agent's capabilities effectively, enabling the identification of vulnerabilities, assessment of system weaknesses, and overall enhancement of the organization's security posture.

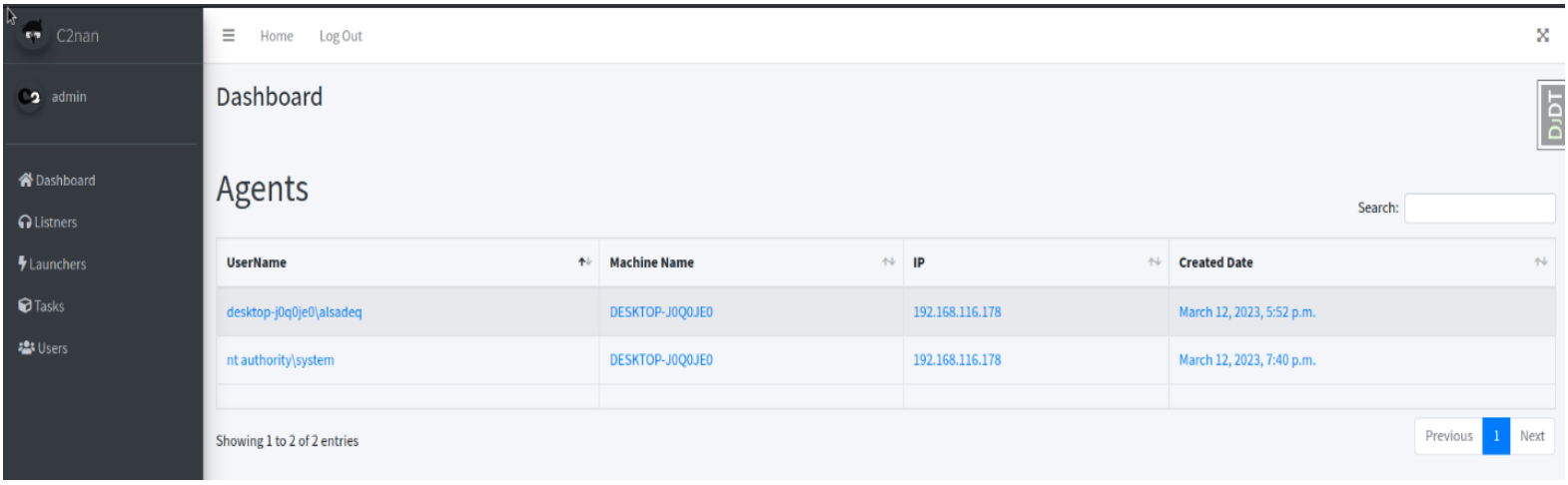

Figure 10 Tool Dashboard

3.1.2 Sending Tasks

Sending tasks play a vital role in a command and control framework within the cybersecurity field. It encompasses the assignment of specific actions or objectives to agents operating within the framework. Tasks can range from conducting vulnerability scans, malware analysis, and log analysis to carrying out incident response actions. Security practitioners, who typically operate the command server, allocate tasks to agents based on the organization's security requirements, priorities, and capabilities of the agents.

Effective task management entails providing clear instructions, setting deadlines, and outlining any associated resources or dependencies. The command-and-control framework should incorporate mechanisms for task tracking, progress monitoring, and reporting to ensure transparency and accountability. By efficiently sending tasks, the framework optimizes resource allocation, streamlines security operations, and empowers security teams to proactively detect and respond to cyber threats systematically.

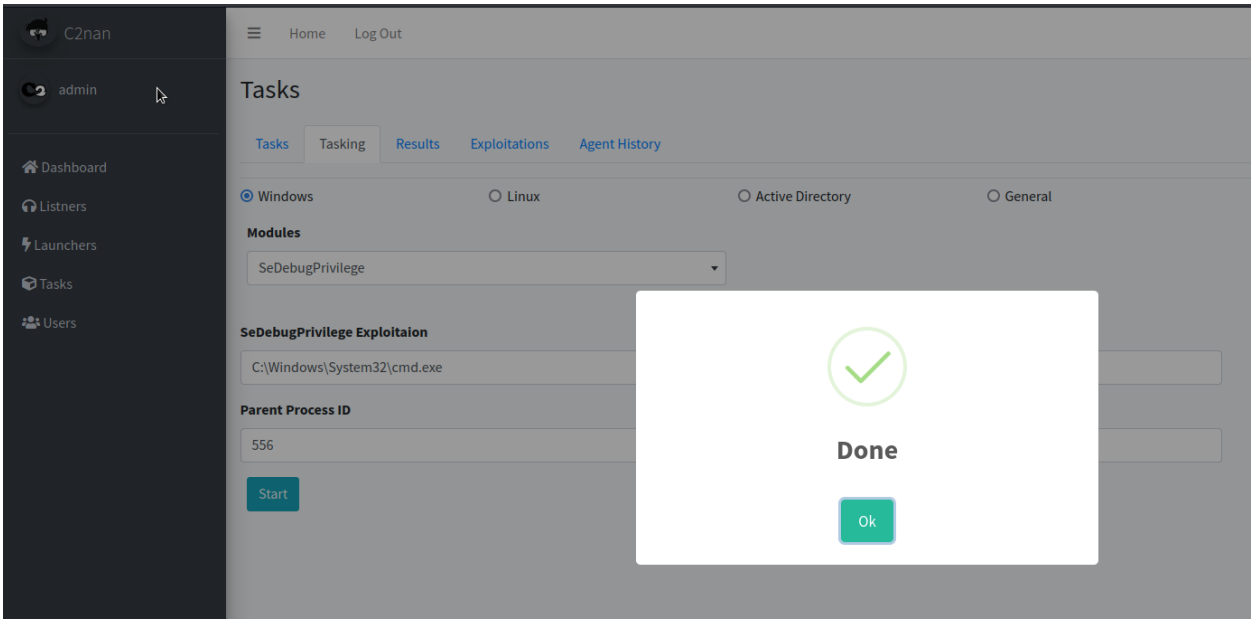

Figure 11 Task Sent Successfully

3.1.3 Provide a report of results

Once agents within the framework have completed their assigned tasks or security assessments, the command server consolidates the collected data, analysis, and findings into comprehensive reports. These reports serve as valuable documentation for security practitioners, management, and stakeholders, providing insights into the organization's security posture, vulnerabilities, and potential risks. The reports typically include a summary of the activities performed,

detected threats, vulnerabilities identified, and recommended mitigation measures.Clear and concise reporting ensures that decision-makers have access to the necessary information to prioritize and allocate resources for remediation efforts. Furthermore, these reports contribute to the overall incident response lifecycle by facilitating incident analysis, post-incident reviews, and compliance audits. By providing thorough and actionable reports, command and control frameworks enhance transparency, support informed decision-making, and enable organizations to continuously improve their cybersecurity defenses.

***Figure* 12 Results View**

3.1.4 Can add to it by advancing automation capabilities for each type of OS

Command and control frameworks hold the potential to enhance automation capabilities across various operating systems (OS) in the cyber security field. By integrating with different OS platforms, these frameworks can leverage automation to streamline security operations and improve overall efficiency. Through scripting and task automation, command and control

frameworks enable the execution of repetitive or complex security tasks across diverse OS environments, including Windows, Linux, macOS, and mobile operating systems.

Automation capabilities encompass areas such as patch management, vulnerability scanning, log analysis, malware detection, and incident response workflows. Automation reduces manual intervention and minimizes the risk of human error, thereby accelerating the speed and accuracy of security operations. This allows security teams to prioritize more strategic and critical tasks. Additionally, command and control frameworks can seamlessly integrate with OS-specific tools and utilities, optimizing the effectiveness of automated processes and ensuring compatibility across different platforms.

The advancement of automation capabilities within command-and-control frameworks empowers organizations to optimize their security efforts and respond efficiently to emerging threats across a range of operating systems.

3.2 Non-Functional Requirements

3.2.1 Familiarizing user interface

Familiarizing oneself with the user interface is crucial to any command-and-control framework within the cybersecurity field. The user interface serves as the primary point of interaction between security practitioners and the framework, enabling them to monitor, analyze, and respond to security events effectively. A well-designed and intuitive user interface plays a vital role in minimizing the learning curve, enabling security teams to swiftly grasp and navigate the framework's functionalities. It should provide clear and organized displays of information, intuitive controls, and customizable views to accommodate individual preferences and specific security requirements. By ensuring user familiarity with the interface, organizations can enhance user productivity, facilitate efficient decision-making, and ultimately fortify their cybersecurity defenses.

3.2.2 Handling multiple agents

As organizations expand in size and complexity, the number of networked devices and endpoints grows, necessitating effective management and coordination of multiple agents within the

framework. These agents, which may include security sensors, intrusion detection systems, and endpoint protection solutions, play a crucial role in gathering and transmitting security information to the central command server. The framework should be designed to handle and communicate with multiple agents efficiently, ensuring seamless data collection, analysis, and response across the network. This entails implementing robust mechanisms for agent

registration, authentication, and task assignment, as well as efficient communication protocols to minimize latency and maximize real-time responsiveness. By proficiently managing multiple agents, command and control frameworks can leverage the network's collective intelligence, enhance situational awareness, and improve the organization's overall security posture.

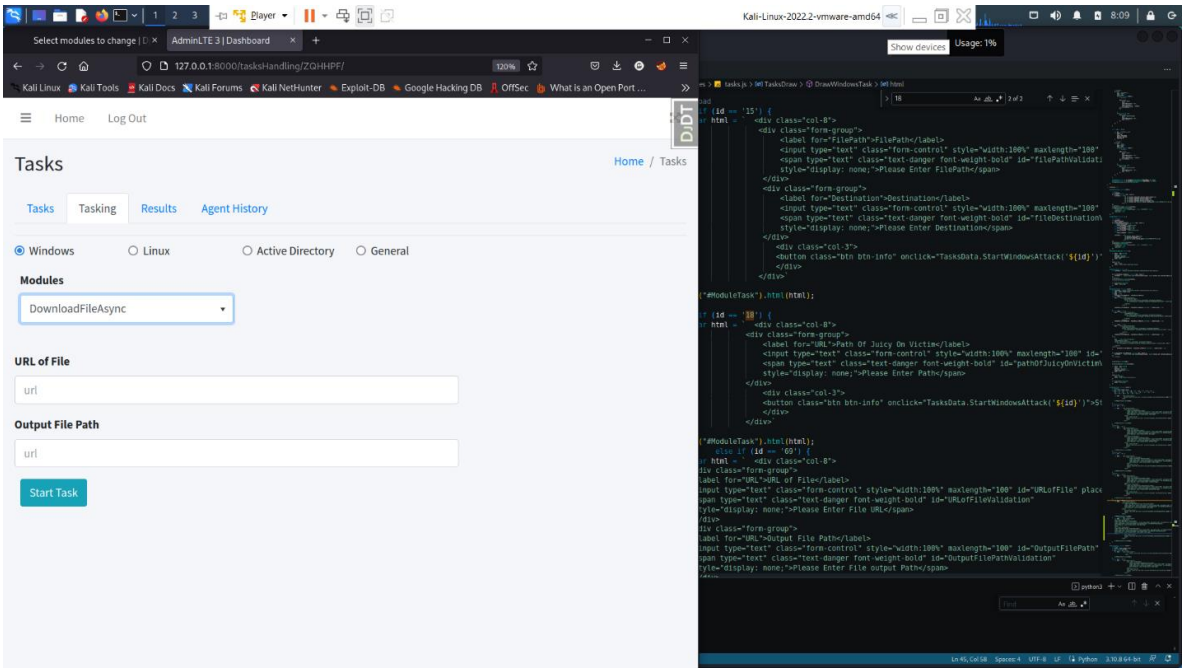

Figure 13 Handling multi-agents.

3.2.3 Providing Multi-User Collaboration

In today's interconnected and collaborative security landscape, organizations often rely on teams of security practitioners working together to detect, analyze, and respond to cyber threats. A command and control framework that supports multi-user collaboration is essential in facilitating seamless communication, information sharing, and task coordination among security team members. It enables the assignment and delegation of tasks, fosters real-time collaboration during incident response, and promotes knowledge sharing and effective decision-making within a secure and controlled environment. Robust access controls and user permissions ensure that only authorized personnel have appropriate levels of access and visibility into sensitive data and actions. By facilitating multi-user collaboration, these frameworks enhance the effectiveness and efficiency of security operations, promoting teamwork and enabling a proactive defense against cyber threats.

3.2.4 Persistent Communication

Persistent communication refers to the continuous and uninterrupted exchange of information between the command server and agents within the framework. This type of communication ensures that real-time updates, commands, and responses are reliably and securely transmitted. It holds particular importance in dynamic and rapidly evolving security environments, where timely information dissemination is crucial for effective threat detection and response. To maintain the integrity, confidentiality, and availability of the exchanged data, robust communication protocols are essential. These protocols include encrypted channels and reliable transport mechanisms. They ensure that data remains protected as it is transmitted between the command server and agents. By enabling persistent communication, command, and control frameworks establish a resilient and responsive network. This allows security practitioners to swiftly adapt to emerging threats, orchestrate coordinated actions, and maintain an up-to-date understanding of the security posture across the organization's infrastructure.

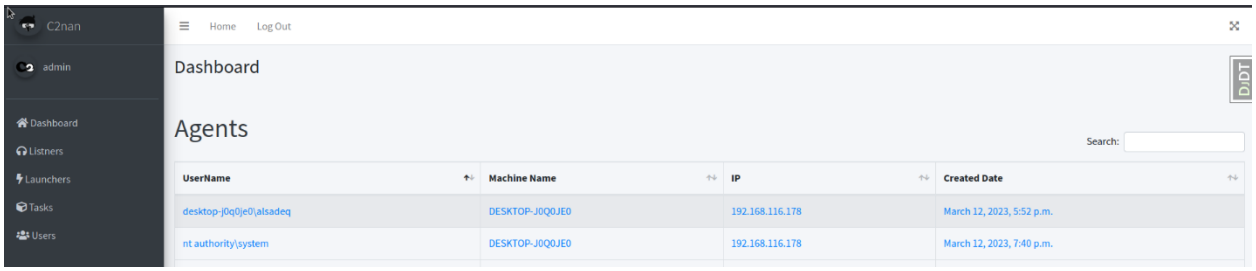

Figure 14 Multi-user Collaboration

Chapter 4
System Architecture & Design

Chapter 4: System Architecture and Design

4.1 Framework

We used the logical design of the covenant.

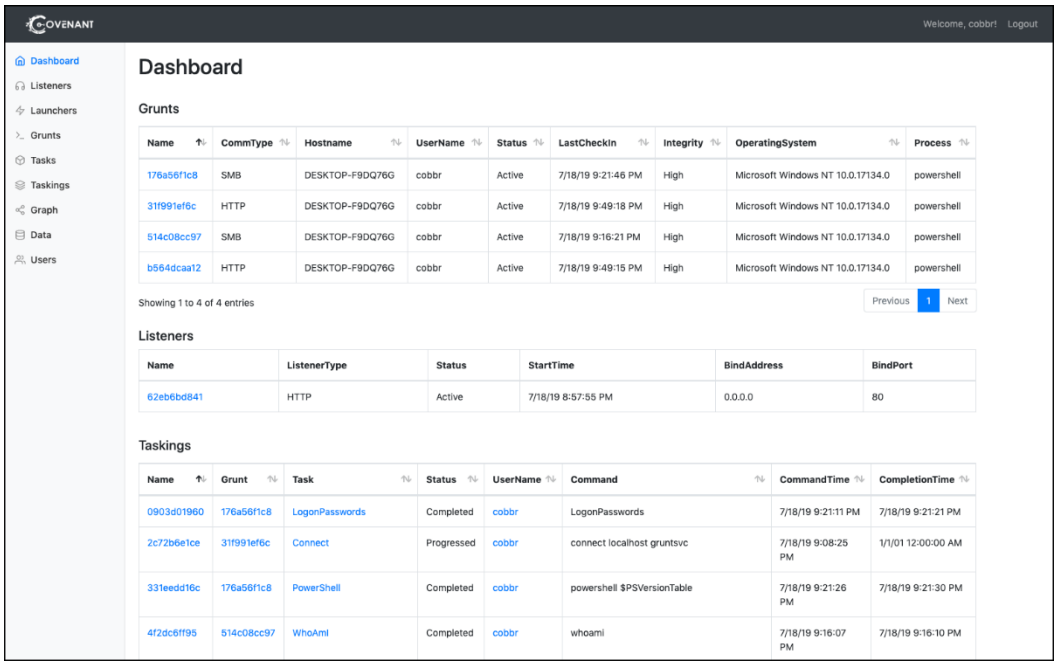

Figure 15 Covenant Dashboard

We have chosen to present our framework through a GUI web interface developed with Django, an open-source web framework that leverages high-level Python. This approach enables developers to concentrate on building their applications without spending excessive time on developing fundamental, clean code. Python, with its OS library, demonstrates compatibility with diverse operating systems. The library provides functions for creating and deleting directories (folders), retrieving their contents, and identifying and modifying the current directory.

We harness this functionality to conduct device audits during the penetration testing process. Our tool is capable of auditing:

- Linux
- Windows
- Active Directory Environment.

By utilizing the robust capabilities of Django and Python, our framework provides a user-friendly interface for security engineers to execute and monitor tasks efficiently.

4.2 Django Architecture

The Django web framework implements a design pattern known as Model-View Template (MVT) architecture.

This architecture separates the user interface, server-side logic, and data models of the web application. And it consists of:

- MODEL: It is the Logical Data Structure. It acts as the intermediate between the Database and View. It manages how and in what format the database is fetched and written, and modified.
- TEMPLATE: is the presentation layer. It looks at how our app looks to the user. It keeps everything that the browser may render, containing HTML and CSS.
- VIEW (Controller): handles Model and Template. It's the brain of the application that controls how data is displayed. It uses Django built-in ORM (object-relational mapper) technique to create a bridge between object-oriented programs and rational databases.

Templates represent the user interface and contain all files that are to be rendered by the server. They interact with the user and take input requests. These input requests are then sent to Views, which processes these requests and responds accordingly. Views can read, write, and modify the database, which is handled by the Model. The Model represents the data and responds to the request sent by the Views. By using the MVT architecture, Django provides a modular, reusable, and maintainable way to develop web applications. This separation of concerns allows for more efficient development, testing, and debugging of the application.

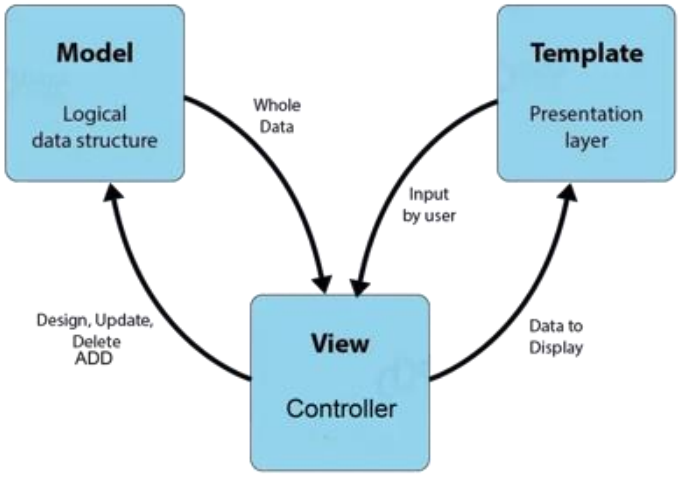

Figure 16 Django Architecture

4.3 Diagrams

4.3.1 Use case Diagram

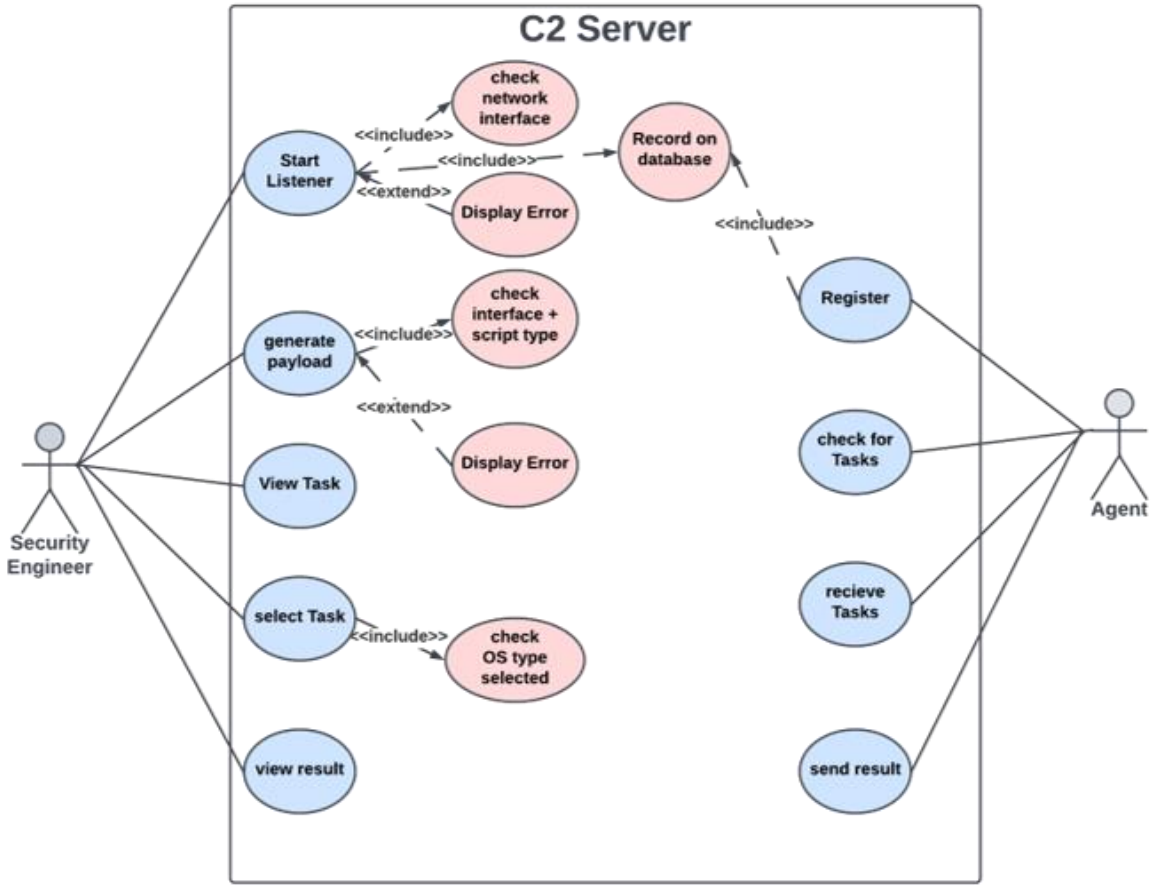

Figure 17 Use case Diagram

By utilizing the robust capabilities of Django and Python, our framework provides a user-friendly interface for security engineers to execute and monitor tasks efficiently. Now, let's delve into the use case diagram of our tool, which encompasses its main functions.

First, an agent can register as a new user, and their relevant data will be securely recorded in the database. This registration process ensures that agents are properly identified and authenticated within the framework.

Next, the security engineer can initiate the listener by utilizing the "start listener" function. Within the diagram, the security engineer is required to enter the correct network interface, such as "eth0". If the provided information is accurate, the relevant data will be recorded in the database. However,

if an incorrect network interface is entered, an error message will prompt the security engineer to input the correct one, ensuring the listener operates smoothly.

Utilizing our tool, the security engineer can generate a payload by selecting the target operating system (Windows or Linux) and inputting the correct network interface information. If any incorrect details are provided, an error message will appear, guiding the security engineer to rectify the input. Once the payload is generated, it can be executed in the command prompt or terminal on the agent's machine, effectively establishing a secure connection between our tool and the agent. The agent will remain active, continuously checking for assigned tasks.

When the security engineer selects a specific task from the available options, they are prompted to input the operating system type. Subsequently, the designated task is sent to the agent, who diligently performs it on their respective system. Once the task is completed, the results are seamlessly transmitted back to our tool, where the security engineer can readily access and review them. This use case diagram showcases the comprehensive functionalities of our tool, providing security engineers with a streamlined workflow for agent management, task execution, and result retrieval.

4.3.2 Sequence Diagram

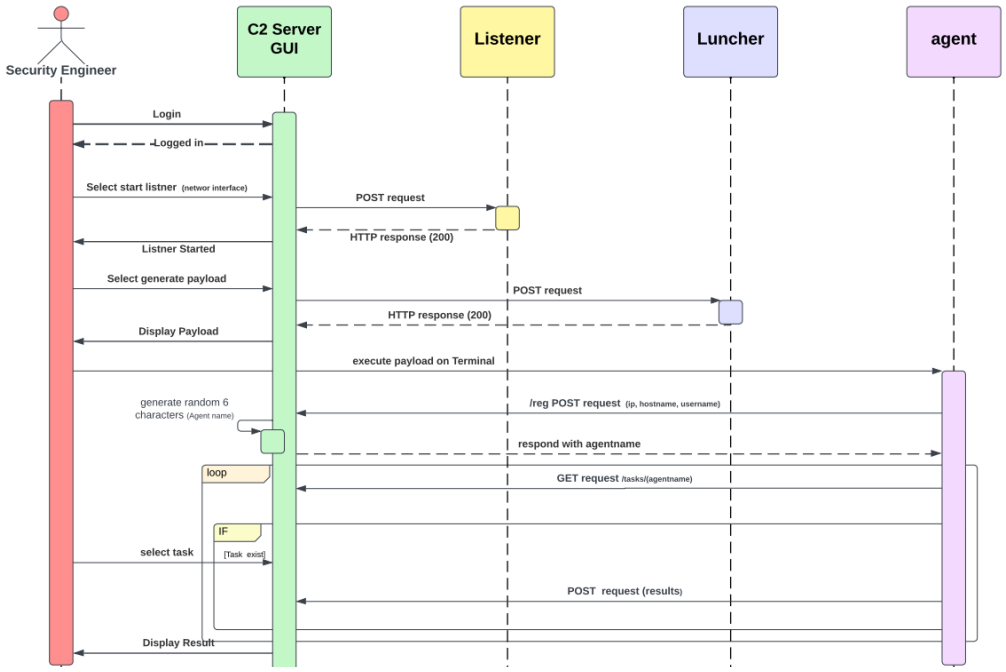

Figure 18 Sequence Diagram

First, the security engineer undergoes the login process to access our tool's GUI (C2 server) after successful validation. Once logged in, the security engineer navigates to the GUI and selects the "start listener" function. They are prompted to enter the specific network interface required for the listener. Subsequently, a post request is sent to the listener function, initiating the listener. An HTTP response with a status code of 200 is returned to the GUI, confirming the successful start of the listener.

Moving forward, the security engineer proceeds to select the "generate payload" option from the GUI. Upon selection, a post request is dispatched to the launcher function. An HTTP response with a status code of 200 is received by the GUI, and the security engineer can now view the generated payload. Subsequently, the security engineer executes the payload within the agent's terminal, initiating the connection between our tool and the agent. As part of this process, the agent sends a post request to our GUI server, providing essential information such as the agent's IP, hostname, and username. The server, upon receiving this data through the "/reg" request, generates a unique six-character "agent name" and responds to the agent with it.

From this stage onward, a continuous loop of communication occurs between our GUI server and the agent. The agent continuously sends a get request to our server with the agent name, specifically targeting the "/task" endpoint. In response, the security engineer selects a task from the GUI, triggering the corresponding task assignment. The updated get request is then relayed back to the agent, which proceeds to execute the assigned task. Once completed, the agent sends a post request containing the task results back to our server. Our server, upon receiving the results, displays them to the security engineer through the GUI.

This loop between our GUI server and the agent ensures seamless communication and task execution, facilitating efficient coordination and monitoring of tasks within the framework. The security engineer can readily view and review the task results, thereby enabling effective analysis and decision-making.

Chapter 5
System Implementation

Chapter 5: System Implementation

Let's talk in detail about each step that happened in the sequence diagram.

5.1 Starting Listener

As described in the sequence diagram, the initiation of the listener requires the security engineer to input the network interface. This crucial step ensures the proper functioning of the listener within our framework. One of the remarkable features of our framework is its persistent communication between the server and agents. This means that even if an agent loses its connection due to a restart or any unforeseen circumstance, once it rejoins the network, the connection is automatically re-established. This seamless reconnection allows the agent to promptly resume its assigned tasks without any disruption. During the implementation phase, we encountered significant challenges when initially setting up individual listeners for each agent on different ports. This approach proved to be error-prone and inefficient. However, through thorough research and analysis, we successfully resolved this issue. We made the necessary modifications by configuring a single listener on port 8000 and directing all agents to connect to it. This enhancement revolutionized the framework's capacity to handle multiple agents concurrently. Now, our framework can seamlessly accommodate and communicate with multiple agents, ensuring smooth and efficient information exchange between the server and the agents. This consolidation of communication channels not only streamlined our implementation but also enhanced the overall performance and scalability of the framework.

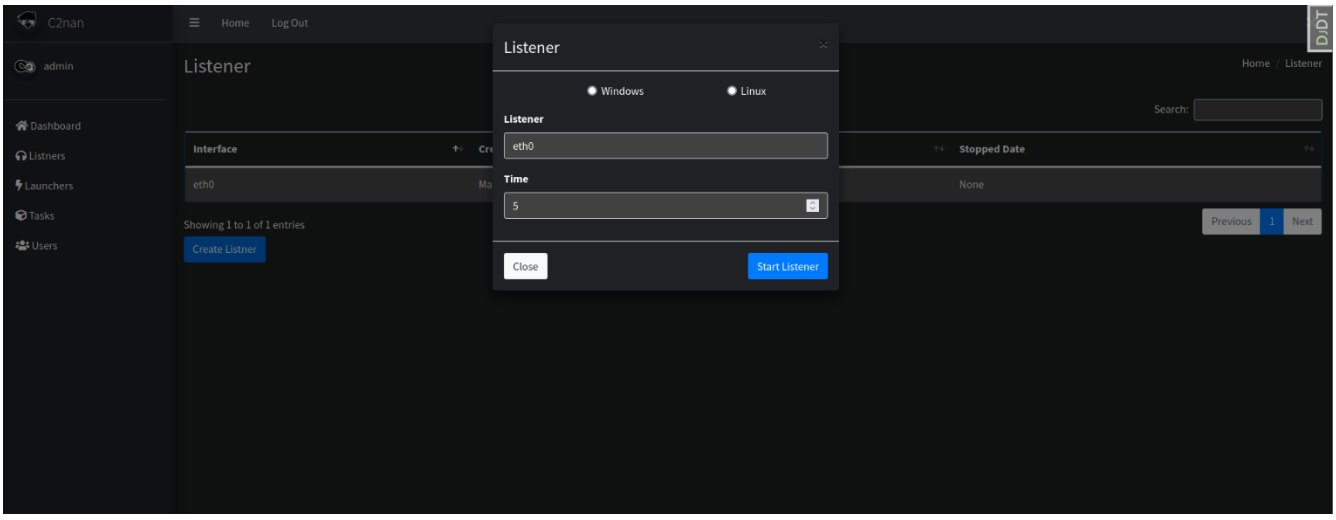

Figure 19 C2nan Listener Page

By addressing these challenges and implementing the necessary modifications, we have successfully established a robust and reliable framework that empowers security engineers to manage and coordinate agents efficiently, facilitating uninterrupted communication and seamless execution of tasks.

5.2 Payload Generator

The payload generator, also known as the launcher, generates a one-liner payload that can be either Bash for Linux or PowerShell for Windows. During the implementation phase, we encountered a significant issue with the PowerShell launcher. When executing a command that contains sub-processes, the launcher would wait for the sub-processes to complete and store the results in a buffer. After completion, the code was then written to extract the results from the buffer and display them. However, this approach caused problems when the executed process was significant, leading to buffer overflow and channel crashes.

```
function RunProcess(FileName, Arguments)
  process = start a new process with FileName as the executable
  process.arguments = Arguments
  redirect process output to an internal buffer
  wait for process to exit
  output = get process standard output from the buffer
  error = get process standard error from the buffer
  return output + error
end function
```

Figure 20 Old power shell pseudo code

To address this issue, we implemented an asynchronous handling method for the results. This was accomplished by storing the finished results in an object and accumulating them in a single variable after the process was completed. The function then returned this variable, which prevented any issues with buffer overflow and channel crashes, ensuring efficient communication between the launcher and the server.

```
function Invoke-executables
    Captures the specified executable stdout, stderr and exit code while they are running
    Setting process invokation parameters
    Creating objects to store stdout and stderr
    Creating string builders to store stdout and stderr
    Starting process
    Retrieving process output
    Accumulate output in results Object
    return results
end of function

put this function in an vbnet ide mode
```

Figure 21 Modified Power shell pseudo code

5.3 Tasks & Results

In our framework, we have considered the unique tasks that can be executed on the agent, considering the specific operating system environment in which the agent operates. Whether it's Linux, Windows, or within an Active Directory environment, each operating system presents distinct capabilities and requirements. To ensure a comprehensive and systematic approach to penetration testing, we have categorized the tasks based on the different phases of testing. These phases encompass key aspects such as Reconnaissance, Enumeration, Vulnerability Analysis, and Exploitation. Within each phase, multiple modules can be executed, addressing specific objectives and techniques relevant to that phase. By classifying the tasks in this manner, our framework provides a structured framework for conducting penetration testing, allowing security engineers to methodically navigate through the various stages and modules. This approach enables a more targeted and comprehensive assessment of the system under evaluation. As the security engineer assigns tasks to agents within the framework, the executed modules generate output that is displayed on the GUI for each agent. This ensures visibility and transparency, allowing security engineers to review the results and analyze the findings effortlessly.

By adopting this classification approach, our framework empowers security engineers with a more structured and comprehensive approach to penetration testing. It facilitates more effective and accurate results, enabling security practitioners to identify vulnerabilities, assess system weaknesses, and ultimately improve the overall security posture of the organization.

Chapter 6
Raspberry Pi Deployment

Chapter 6: Raspberry Pi Deployment

6.1 Hardware Requirements

The Raspberry Pi 4 with 4GB of RAM has several hardware requirements that must be met to run properly. These requirements include:

- Power supply: The Raspberry Pi 4 requires a 5V DC power supply with a minimum current rating of 3A. A power supply that does not meet this requirement can cause the device to malfunction or not start up at all.
- MicroSD card: The Raspberry Pi 4 does not have built-in storage, so a microSD card is required to run the operating system and store data. A microSD card with a minimum capacity of 8GB is recommended, although larger cards can be used if needed.
- HDMI display: The Raspberry Pi 4 supports dual monitor displays at resolutions up to 4K, but at least one HDMI display is required for setup and initial use.
- USB keyboard and mouse: A USB keyboard and mouse are required to navigate the Raspberry Pi 4's operating system and perform basic functions.
- Cooling: The Raspberry Pi 4 can generate a significant amount of heat, especially when under heavy load, so it is recommended to use a case with built-in cooling or to add a cooling fan or heat sink to prevent overheating.
- Network connectivity: The Raspberry Pi 4 has built-in Wi-Fi and Bluetooth, but a wired Ethernet connection is recommended for best performance.

The Raspberry Pi 4 with 4GB of RAM is a powerful device that can handle a wide variety of tasks, but it does have some specific hardware requirements that must be met to operate effectively.

6.2 Software Requirements

The Raspberry Pi 4 with 4GB of RAM has specific software requirements that must be met to run properly. These requirements include:

Operating system: The Raspberry Pi 4 can run a variety of operating systems, but the most common one is Raspberry Pi OS (formerly known as Raspbian), which is a Linux-based operating system designed specifically for Raspberry Pi. Other operating systems that are compatible with the Raspberry Pi 4 include Ubuntu, Debian, and various other Linux distributions.

Software updates: Regular software updates are important for keeping the Raspberry Pi 4 running smoothly and securely. The operating system and any installed software should be updated regularly using the appropriate update commands.
Development tools: If the Raspberry Pi 4 is being used for development purposes, development tools such as Python, Java, and other programming languages and libraries may need to be installed.
Applications: The Raspberry Pi 4 can run a wide variety of applications, from web browsers and office suites to media players and games. However, the specific applications required will depend on the intended use of the device.
Drivers: Depending on the hardware peripherals that are connected to the Raspberry Pi 4, additional drivers may need to be installed to ensure proper operation.

The Raspberry Pi 4 with 4GB of RAM has relatively straightforward software requirements, as it can run a wide variety of operating systems and applications. Regular updates and proper installation of drivers and development tools are important for ensuring that the device runs smoothly and efficiently.

6.3 Deployment Steps

6.3.1 Why should we choose rpi4 model B?

Initially, we considered using something to deploy our project on. In some companies, the blue team may not know that you are here, so you may be able to enter with a laptop. However, some companies may prevent you from entering with your laptop. In these cases, having a mobile tool such as a Raspberry Pi that is already deployed can be useful. Therefore, we chose to use the Raspberry Pi, and for any improvements to our tool, we decided to use the Raspberry Pi 4 B with 4GB RAM. It currently meets our needs and will also help us in the future with added features.

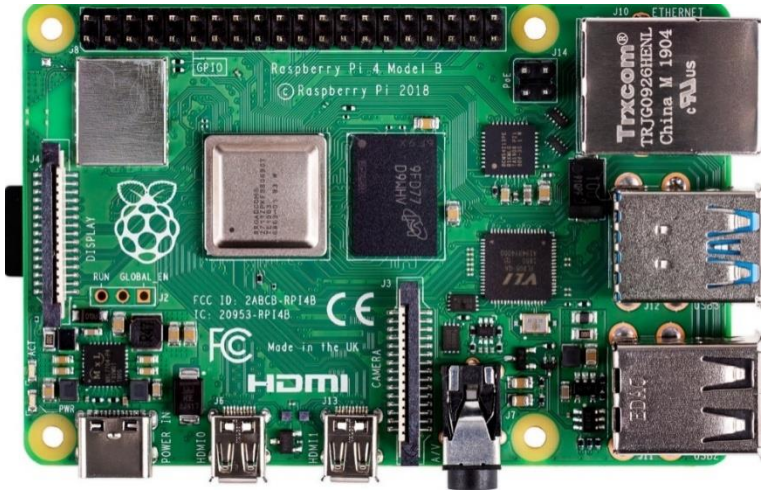

Figure 22 Raspberry pi 4

6.3.2 Setup of our RPI

First, we started by installing the operating system, and we chose Linux. Specifically, we decided to use Kali Linux as it will be helpful for our project:

Kali Linux is a Debian-based Linux distribution that is designed for digital forensics and penetration testing. It is available for many different platforms, including the Raspberry Pi 4 Model B. Here are some key features of Kali Linux on the Raspberry Pi 4 Model B:

- Pre-installed tools: Kali Linux comes with a variety of pre-installed tools for penetration testing and digital forensics, including Metasploit, Nmap, and Wireshark.
- ARM architecture support: Kali Linux is optimized to run on ARM-based devices, including the Raspberry Pi 4 Model B, so it can take full advantage of the device's hardware capabilities.
- Easy installation: Kali Linux can be installed on the Raspberry Pi 4 Model B using a simple command-line installer or by using pre-built images that can be written to an SD card.
- Customization: Kali Linux is highly customizable, allowing users to add or remove packages and configure the system to meet their specific needs.
- Community support: Kali Linux has a large community of users and developers who contribute to the project, provide support, and develop new tools and features.

  Kali Linux is a powerful and flexible operating system that can be a great choice for security professionals, researchers, and enthusiasts who want to use the Raspberry Pi 4 Model B for penetration testing and digital forensics.

After setting up the operating system, we began to consider the various ways we could connect the Raspberry Pi with a computer in the company, which we referred to as the 'agent.' There are several options available to us:

- SSH (Secure Shell): This is a command-line tool that allows you to remotely access and control your Raspberry Pi from another computer. It comes pre-installed on most Raspberry Pi operating systems.
- VNC (Virtual Network Computing): This is a graphical desktop sharing system that allows you to remotely access your Raspberry Pi's desktop from another computer. You will need to install a VNC server on your Raspberry Pi and a VNC client on your computer.
- Remote Desktop: If you are using a Windows computer, you can use the built-in Remote Desktop feature to remotely access your Raspberry Pi's desktop. You will need to enable Remote Desktop on your Raspberry Pi and configure your firewall settings.
- SFTP (Secure File Transfer Protocol): This is a secure way to transfer files between your Raspberry Pi and computer. You can use an SFTP client such as FileZilla to transfer files between your devices.
- SSHFS (SSH File System): This allows you to mount your Raspberry Pi's file system on your computer over an SSH connection. You will need to install SSHFS on both your Raspberry Pi and computer.

These applications provide convenient ways to connect and control your Raspberry Pi from a computer, allowing you to access files, run commands, and perform tasks remotely. After exploring our options, we found that several applications could achieve this goal. However, we determined that the most effective and secure method would be to use SSH.

SSH:

- SSH (Secure Shell) is a network protocol that allows secure communication between two computers over an unsecured network. It provides a secure and encrypted way to remotely access and control a computer, as well as transfer files between computers.
- SSH operates through a client-server model, where the SSH client is used to initiate a connection to an SSH server. Once connected, the user can securely execute commands on the remote server or transfer files between the two computers.

- One of the primary benefits of SSH is its security features. SSH encrypts all data exchanged between the client and server, including passwords and commands, which helps prevent unauthorized access and interception of sensitive information. Additionally, SSH uses public-key cryptography to authenticate the server and client, ensuring that both parties can trust each other's identities.
- SSH is commonly used by system administrators, developers, and other IT professionals to remotely manage servers and network devices, as well as to securely transfer files. SSH is also often used in conjunction with other protocols, such as SFTP (Secure File Transfer Protocol) and SCP (Secure Copy), to provide additional security for file transfers.

Figure 23 Secure shell connection (SSH)

- SSH is a powerful and versatile tool for securely accessing and controlling remote computers, and its widespread use and support make it a reliable choice for many applications.
- After discussing SSH, it's time to use the Raspberry Pi (RPI). We can connect the RPI to a computer using SSH and then do what we want on it. First, we need to install our tool on it and set up the database we will use. This will require us to install XAMP for Windows or LAMP for Linux. Since we are using Kali Linux, we choose LAMP. However, we encountered a problem during installation, so we changed our minds and decided to install each component separately. We installed each component separately and tested each one. Now it works. After that, we put our database on phpMyAdmin and started to run the project on the RPI. We

encountered a lot of errors, some related to libraries and others new errors. We tried to fix all of them, and it took a lot of time, but now it works well.

6.4 Security Issues

We know that we have more than will problems with security issues. Now will discuss them and some solutions for these problems:

6.4.1 Port security meaning

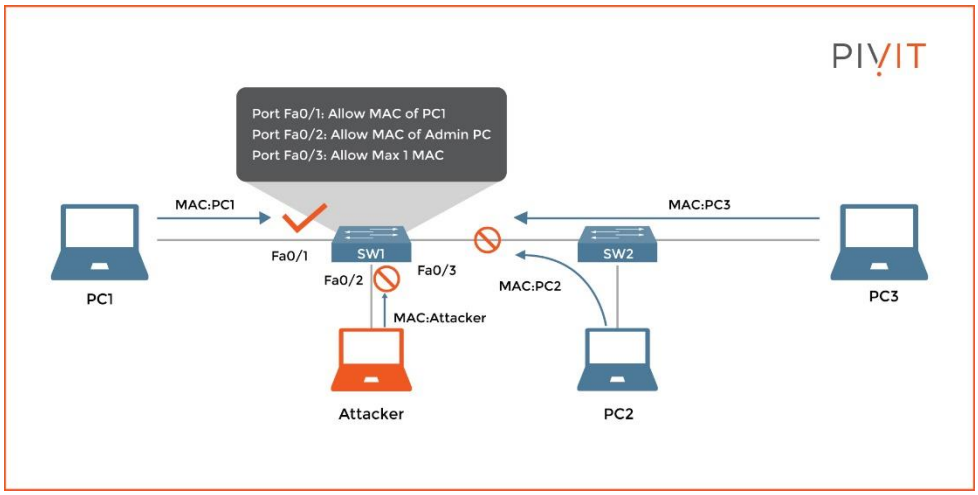

Figure 24 Port Security idea

Port security is a security feature used in networking to restrict access to a network port to only authorized devices. It is typically used in switches and routers to control access to a network and prevent unauthorized devices from connecting and accessing the network. Port security can be configured in several ways, including limiting the number of devices that can connect to a port, limiting access based on MAC address, and disabling a port if a security violation occurs. The goal of port security is to protect the network from unauthorized access and prevent network security breaches. For example, in a corporate network, port security can be used to restrict access to a switch port to only the devices that are associated with that port. This ensures that no unauthorized devices can connect to the network and potentially compromise its security.

Port security is an important security feature that helps to secure networks and protect against unauthorized access.

6.4.2 How port security works:

MAC Address Limiting: The switch or router is configured to only allow a specific number of MAC addresses to connect to a particular port. This makes it difficult for unauthorized devices to connect to the network as they will not have a MAC address that is on the allowed list.

Port Disablement: Unused or unnecessary ports can be disabled to reduce the attack surface of the network. This makes it difficult for unauthorized devices to connect to the network as there will be no open ports available.

Port-Based Network Access Control (PNAC): PNAC can be used to restrict access to specific users or groups. This is achieved by configuring the switch or router to only allow specific devices to connect to a particular port based on the device's MAC address or IP address.

Sticky MAC Addresses: This feature allows the switch or router to dynamically learn and configure a list of allowed MAC addresses for a particular port. If an unauthorized device tries to connect to the network, it will be prevented from doing so, as its MAC address will not be on the allowed list.

DHCP Snooping: This feature helps to prevent rogue DHCP servers from being connected to the network. A rogue DHCP server can assign incorrect IP addresses to network devices, causing network connectivity issues.

Dynamic ARP Inspection (DAI): This feature helps to prevent ARP spoofing attacks, where an attacker sends false ARP messages to map their MAC address to the IP address of another device on the network.

IP Source Guard: This feature restricts IP traffic on an untrusted LAN port by only allowing traffic from IP addresses that have been dynamically learned or configured by the administrator.

802.1X Authentication: This feature provides a more secure method of authentication for network devices. Devices are required to present a valid certificate or other form of authentication before they are allowed to connect to the network.

Port Security Violation Actions: In the event of a port security violation, the switch or router can be configured to take specific actions, such as disabling the port, sending an alert, or logging the event.

These are just some of the measures used to enhance port security. The exact set of measures will depend on the specific requirements of the network, and it is important to regularly review and update port security measures to stay ahead of evolving security threats.

Internal network

An internal network, also known as an intranet, is a private network that is used within an organization. It is typically used to share resources, such as files, printers, and applications, among the employees of an organization.

An internal network is distinct from the Internet, which is a public network that is used to connect computers and other devices around the world. An internal network is protected by firewalls and other security measures to ensure that only authorized users can access the network and its resources.

Internal networks are commonly used in offices, schools, and other organizations to provide a secure and convenient way for employees to access the resources they need to do their jobs. They can also be used to create virtual private networks (VPNs), which allow employees to securely access the internal network from remote locations.

And we started to search to find something to help us to skip these problems that we, got then we found more than one way.

➢ **Spoof:**

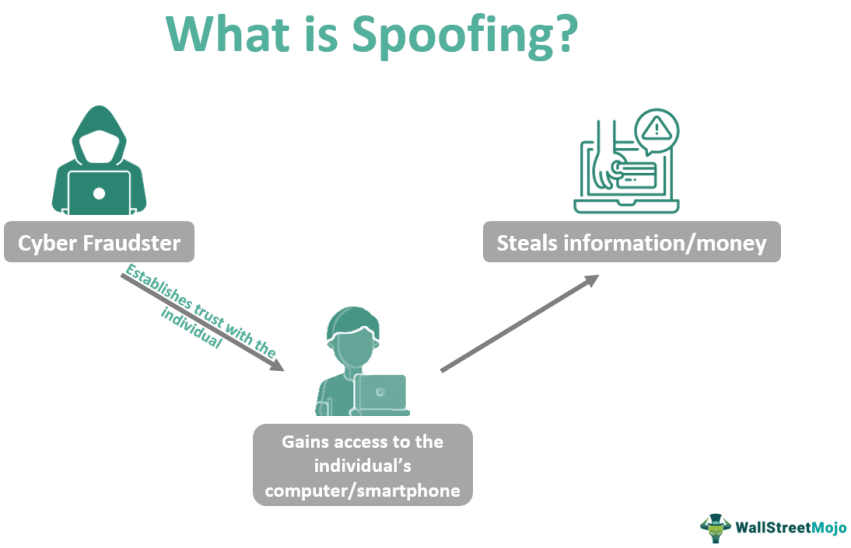

Figure 25 Spoofing

Spoofing is a technique used to deceive or trick a system or user into believing that something is legitimate when it is not. In the context of computer security, spoofing refers to various methods used to hide the true identity of a user, device, or system and make it appear as if it is something else.

MAC Spoofing: This involves changing the Media Access Control (MAC) address of a device to make it appear as if it is another device. This can be used to bypass network security controls that restrict access based on MAC address.

IP Spoofing: This involves altering the source IP address of a network packet to make it appear as if it came from a different device. This is often used in network attacks to hide the identity of the attacker.

Email Spoofing: This involves altering the "From" field in an email to make it appear as if it came from a different sender. This is often used in phishing attacks to trick users into giving up sensitive information.

DNS Spoofing: This involves altering the domain name system (DNS) response to redirect a user to a fake website, even though they intend to visit a different website.

URL Spoofing: This involves creating a fake URL that appears similar to a legitimate URL and tricking users into visiting the fake website and entering sensitive information.

There are many techniques similar to spoofing that can be used to deceive or trick systems and users, including:

Man-in-the-Middle (MitM) Attack: This involves intercepting communications between two parties, allowing the attacker to eavesdrop on or modify the data being exchanged. This can be done through techniques such as ARP spoofing, where an attacker changes the ARP cache on a target device to redirect traffic to the attacker's device.

Phishing: This involves tricking users into revealing sensitive information, such as passwords or credit card numbers, by posing as a trusted entity. This is often done through fake emails or websites that appear to be from a legitimate source.

Smishing: This is a type of phishing that uses text messages instead of emails to trick users into revealing sensitive information.

Malware: This is a type of software that is designed to cause harm to systems or steal sensitive information. Malware can be delivered through email attachments, infected websites, or infected software downloads.

Impersonation: This involves pretending to be someone else, either in person or online, to gain access to sensitive information or resources.

These are just a few examples of techniques that are similar to spoofing.

Chapter 7
Evasion Techniques

Chapter 7: Evasion Techniques

7.1 Overview of Evasion Techniques

Evasion techniques refer to the methods used to bypass or evade security measures put in place to detect or prevent attacks. These techniques are used by attackers to hide malicious activities, avoid detection, and increase the chances of successful attacks.

Evasion techniques can be used at different stages of an attack, including during reconnaissance, exploitation, and post-exploitation phases. Attackers use different methods to evade detection, such as obfuscation, encryption, fragmentation, and tunneling.

7.2 Types of Evasion Techniques

There are several types of evasion techniques that attackers use to bypass security measures and avoid detection. Here are **some** of the common types of evasion techniques:

- **Obfuscation**: Obfuscation involves the use of various techniques to **hide** the true nature of malicious code or data. This may involve altering the code's structure, changing variable names, or using encryption to hide the code's content.
- **Encryption**: Encryption involves encoding data so that it can only be read by authorized parties who have the decryption key. Attackers use encryption to hide the contents of malicious traffic or to prevent security devices from detecting malicious traffic.
- **Fragmentation**: Fragmentation involves splitting malicious traffic into smaller packets or segments. This makes it harder for security devices to detect malicious traffic as the packets may arrive at different times or from dissimilar sources.
- **Protocol-level Evasion**: Protocol-level evasion involves exploiting weaknesses in the protocols used by network devices to hide malicious traffic. For example, attackers can use TCP/IP protocol evasion techniques to bypass firewalls and intrusion detection systems.
- **Timing-based Evasion**: Timing-based evasion involves launching attacks at specific times or intervals to evade detection. This can involve launching attacks during periods of high network activity or using random time intervals to avoid detection.
- **Traffic-level Evasion**: Traffic-level evasion involves using techniques to manipulate network traffic to avoid detection. For example, attackers can use tunneling or packet injection techniques to hide malicious traffic within legitimate traffic.

- **Signature Evasion**: Signature evasion involves modifying or obfuscating the signature of a known attack to avoid detection by security devices that rely on signature-based detection.

These are just some of the common types of evasion techniques that attackers use. As attackers become more sophisticated, they are constantly developing new techniques to bypass security measures, which is why it is important for organizations to stay up to date with the latest threats and have strong security measures in place to protect against attacks.

7.3. Implementation Details

First, let's talk about an important topic in this chapter: the first evasion technique that we have already added to our project AMSI bypass.

7.3.1 AMSI (Antimalware Scan Interface)

is a Microsoft security feature introduced in Windows 10 and Windows Server 2016 that enables software to be scanned for malware at runtime. The purpose of AMSI is to provide an additional layer of protection against malware by enabling applications to be scanned for malware before execution.

However, as with many security features, attackers have found ways to bypass AMSI to evade detection. There are various techniques for bypassing AMSI, including:

- Code Obfuscation: By obfuscating the malicious code, attackers can make it more difficult for AMSI to detect the malware.
- Payload Encryption: By encrypting the payload, attackers can make it more difficult for AMSI to detect the malware.
- AMSI Unhooking: Attackers can disable AMSI by unhooking the AMSI DLL from the targeted process.
- DLL Injection: Attackers can inject a DLL into a process, allowing them to bypass AMSI by intercepting and modifying its function calls.
- Patching: Attackers can patch the AMSI DLL or the code calling it to prevent detection.

7.3.2 AMSI bypass

If you use an AMSI bypass, you can bypass attack or filtration mechanisms. This allows you to make a bypass for any interface, session, or command line where you can enter a one-liner malicious command. This technique is considered an evasion technique, and detection is difficult because AMSI has already been bypassed.
Second, let's talk about another important topic in this chapter: the second evasion technique that we have already added to our project bypass certificate validation.

7.3.3 Certificate validation

Certificate validation is the process of verifying the authenticity and validity of a digital certificate. Digital certificates are used to ensure secure communication over the internet by verifying the identity of the parties involved in the communication.
When a user connects to a website or other online service, the service presents a digital certificate that contains information about its identity, including its public key. The user's computer then checks the certificate's digital signature to verify its authenticity, and also checks that the certificate has not been revoked or expired.
If the certificate is deemed valid, the user's computer and the server can establish a secure connection using encryption techniques such as SSL/TLS. However, if the certificate is found to be invalid or suspicious, the connection may be blocked, or the user may be warned of potential security risks.

7.3.4 Types of certificate validation

- Domain Validation (DV): This type of validation checks if the certificate holder has control over the domain name. It is the most basic level of validation.
- Organization Validation (OV): This type of validation includes checking the organization's legal existence, physical address, and domain name control. It provides a higher level of trust than DV.

- Extended Validation (EV): This type of validation is the most extensive and includes a thorough background check of the organization. It provides the highest level of trust, and the web browser displays a green padlock and company name in the address bar.
- Certificate Transparency (CT): This type of validation is a public log of all certificates issued by Certificate Authorities (CA). It allows website owners to monitor for unauthorized certificate issuance.
- Certificate Pinning: This technique allows website owners to specify which certificate authorities are authorized to issue certificates for their domains. This helps prevent fraudulent certificates from being issued by rogue CAs.

7.3.5 Certificate validation bypass

To encrypt the traffic of our tool, we first used HTTPS. However, as we know, HTTPS requires a certificate, we couldn't obtain a valid one, so we generated our own certificate to be able to start the HTTPS technique, however it's not trusted by any other user since it's just a local certificate. Therefore, we created a second evasion technique, **the certificate validation bypass**, which can be done in two ways. First, we edited our **one-liner** command to bypass the certificate validation during the request. Second, we needed to bypass the certificate validation from the **agent's side** since it was always in a loop and continuously sending and receiving requests.

- **Obfuscation**:

Obfuscation refers to the practice of deliberately making software or code difficult to understand or analyze. The goal of obfuscation is to prevent attackers from reverse engineering or understanding the underlying logic of a program, thereby making it harder for them to exploit vulnerabilities or discover sensitive information.

Obfuscation techniques can include things like renaming variables or functions, adding unnecessary code or comments, encrypting, or compressing code, and using code obfuscation tools. By applying obfuscation techniques, the original code is made more complex and difficult to understand, making it harder for attackers to reverse engineer it and find vulnerabilities. Obfuscation can be used in various areas of computer security, including malware development, software licensing, and protecting sensitive data. However, it should be noted that obfuscation is not a foolproof defense mechanism and can be bypassed by determined attackers who have access to the right tools and techniques.

- **Our processes:**

During our control there're tasks and results files placed in the agent OS; we have made our controlling of the system as stealthy as possible in our C2 framework. Specifically, we have used names and paths that closely resemble default files in the Windows or Linux operating system, which makes it less likely for a normal user to notice any unusual activity on the system. Additionally, we have stored these files in a hidden format that the agent would not typically access or work with during its routine tasks. By using obfuscation techniques, we have made it more difficult for security systems and users to detect any suspicious activity on the system, allowing us to maintain our control and carry out our malicious activities undetected.

Chapter 8
Testing & Evaluation

Chapter 8: Testing and Evaluation

In this section, we will address specific test cases that are crucial for the tool's functionality, considering the intended user as a **security engineer** rather than an average user. To begin, we have conducted thorough testing on the login page, ensuring appropriate validations are in place to prevent unauthorized access. Additionally, each button on the home page has been extensively tested to verify its proper functionality.

Moving forward, our focus will now shift to testing the four key components of the C2nan tool.

8.1 Listener Testing

When the listener section is accessed, and the option to create a listener is chosen, default inputs are presented to facilitate seamless interactions with GUI. These default inputs serve as convenient starting points, offering examples and guidelines to assist the security engineer in utilizing the tool effectively. Furthermore, it prevents errors and minimizes the likelihood of incorrect or invalid input from the security engineer.

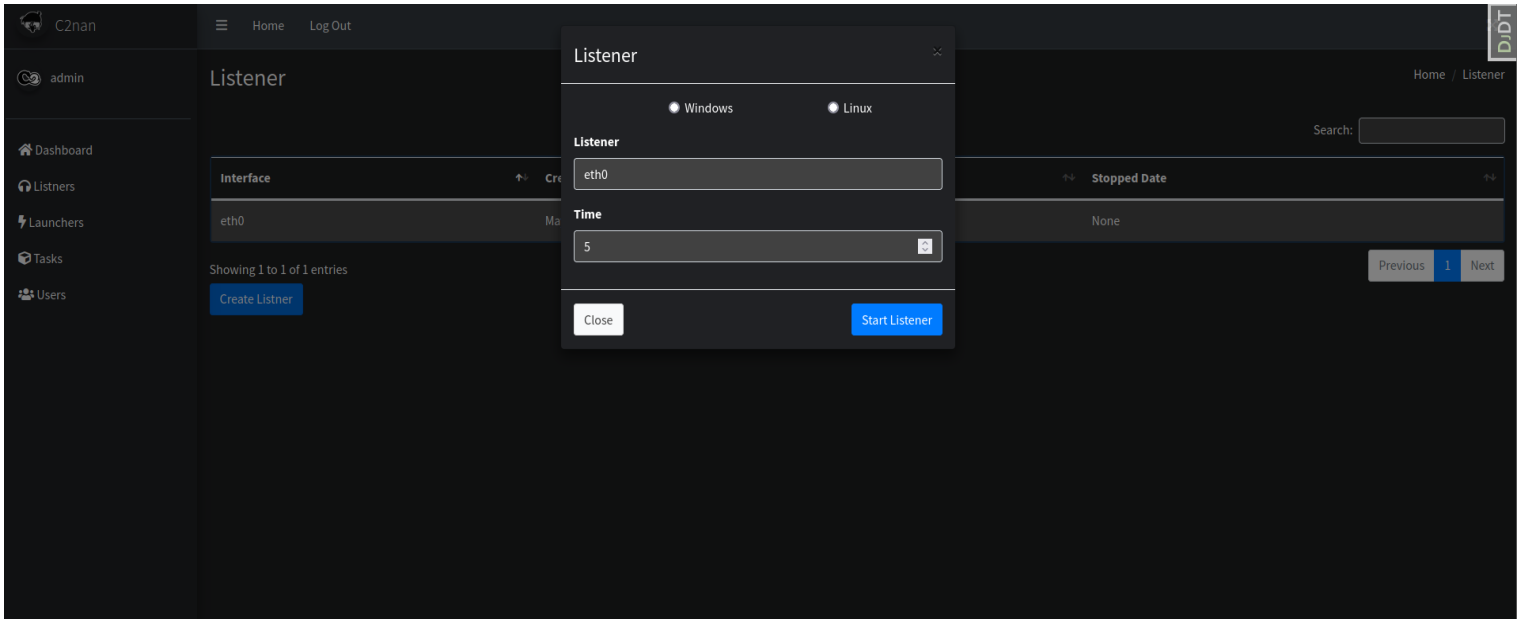

Figure 26 Listener Creation Default Page

➢ Empty input Case:

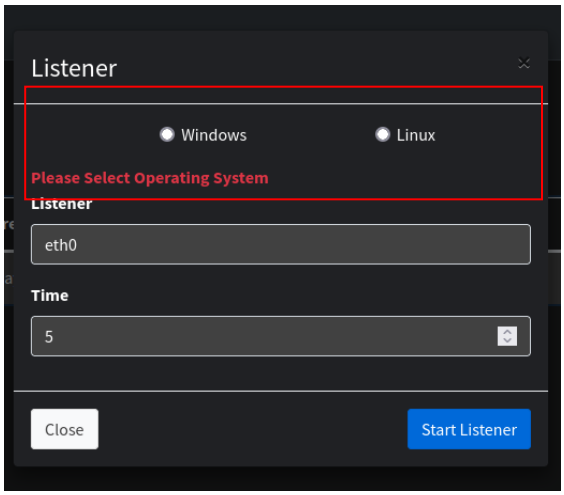

Figure 28 OS validation

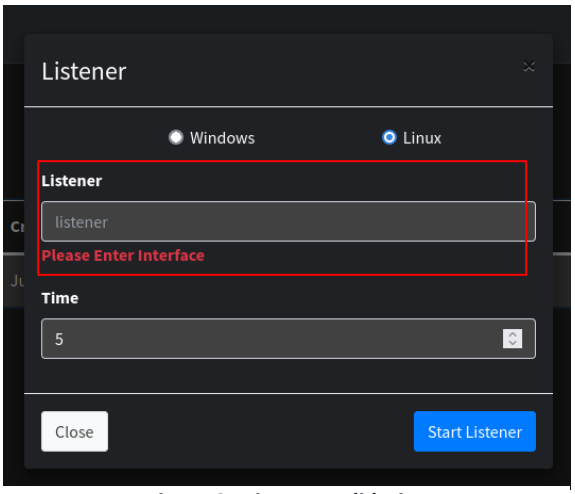

Figure 27 Listener validation

The listener creation process includes robust validations to ensure the selection of the operating system (OS) and network interface. These validations are designed to enforce the requirement that both an **OS type** and a **network interface** must be specified to create a listener successfully.

If the user does not provide a value for the **"jitter"** input (time input as displayed in the GUI), a default value of **5** is automatically assigned. This means that, by default, the agent will send a GET request to the server **every 5 seconds.**

➢ Enter Un-existing Interface Case:

In situations where a non-existent network interface is entered, such as specifying "tun0" while the only functioning interface is "eth0," a specific test case arises. Although the GUI displays a **"save failed"** message, upon examining the dashboard, it becomes apparent that the listener is saved in the listeners' table. Consequently, this test case is considered a **failure**.

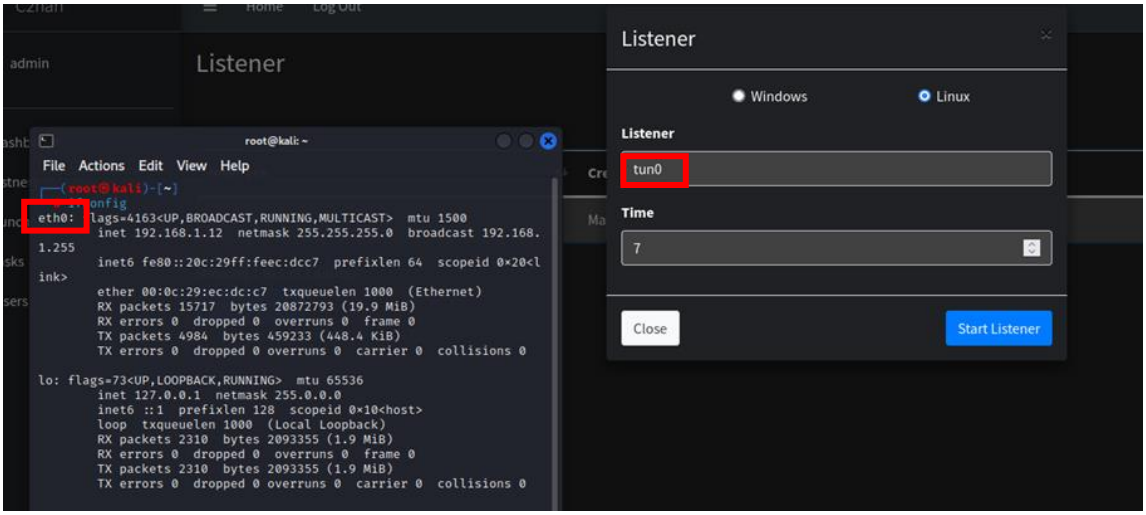

Figure 29 Un-existing interface test case

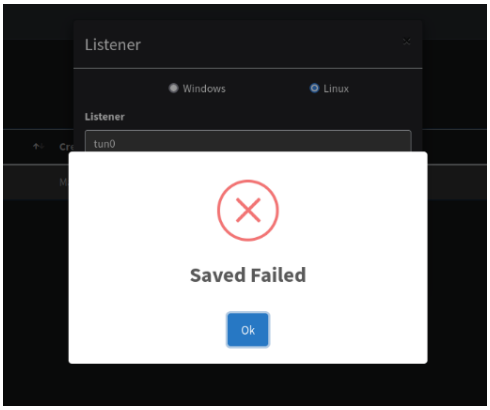

Figure 30 Error message in listener creation

This discrepancy between the GUI message and the listener's actual persistence in the table highlights a potential issue in the tool's validation process. This specific test case demonstrates the need for improvements in the tool's validation process to provide consistent and reliable feedback to security engineers.

- **Duplicate the interface Case:**

In scenarios where multiple listeners are created using the same interface, an interesting observation arises regarding their appearance in the listener table. When more than one listener is associated with a non-eth0 interface, duplicates of the listeners are displayed in the listener table. However, when it comes to the eth0 interface, no duplication occurs. And this indicates a small issue in the validations on this point as it works for the input “eth0” where the table is updated instead of duplication; this issue is now **solved** after checking this test case.

Figure 31 Duplicating Interface

➢ Successful creation Case:

This test case ensures that the system can handle correctly entered inputs. The system validates the inputs and processes them accordingly. And then, the GUI displays a success message **"Listener created successfully."**

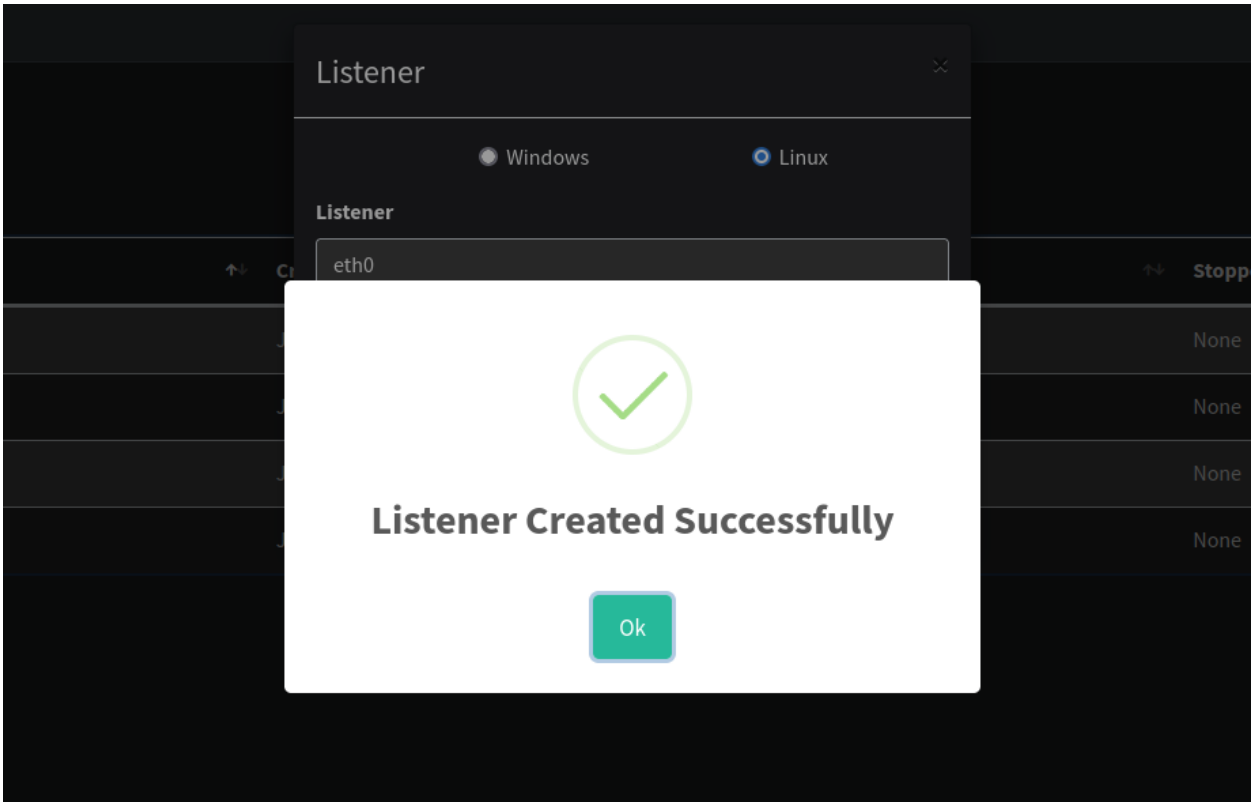

Figure 32 Listener Successful Creation

After creating the listener, the system automatically generates the payload and displays it on the GUI listener screen.

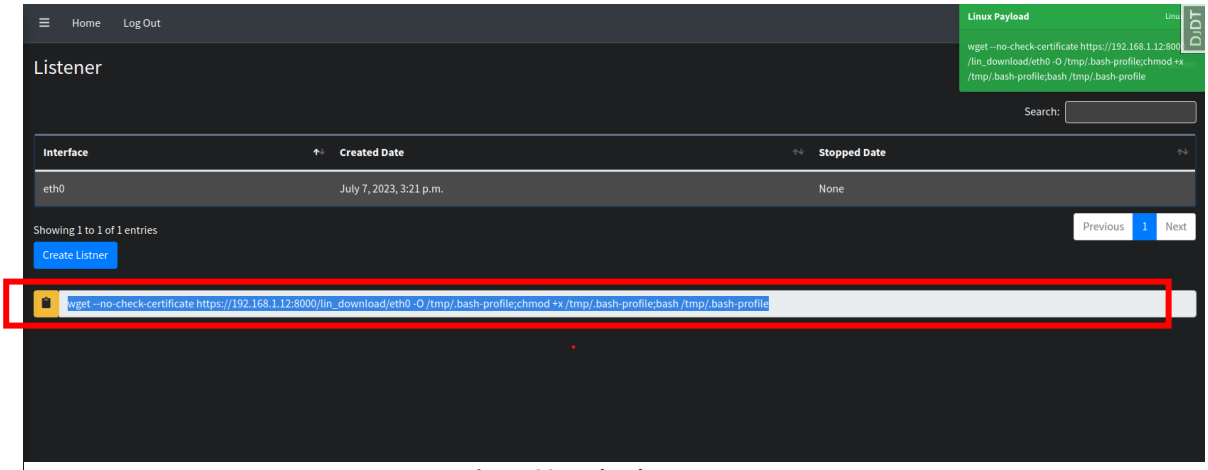

Figure 33 Payload Appearance

8.2 Payload Generator Testing

On the launcher page, security engineers are presented with a mandatory choice between two options: bash and Power shell. The selection made here is essential, and the tool incorporates validations to ensure that a choice is always made. By incorporating these validations, the tool eliminates the possibility of launching without a defined command interpreter, providing a clear and straightforward user experience for security engineers.

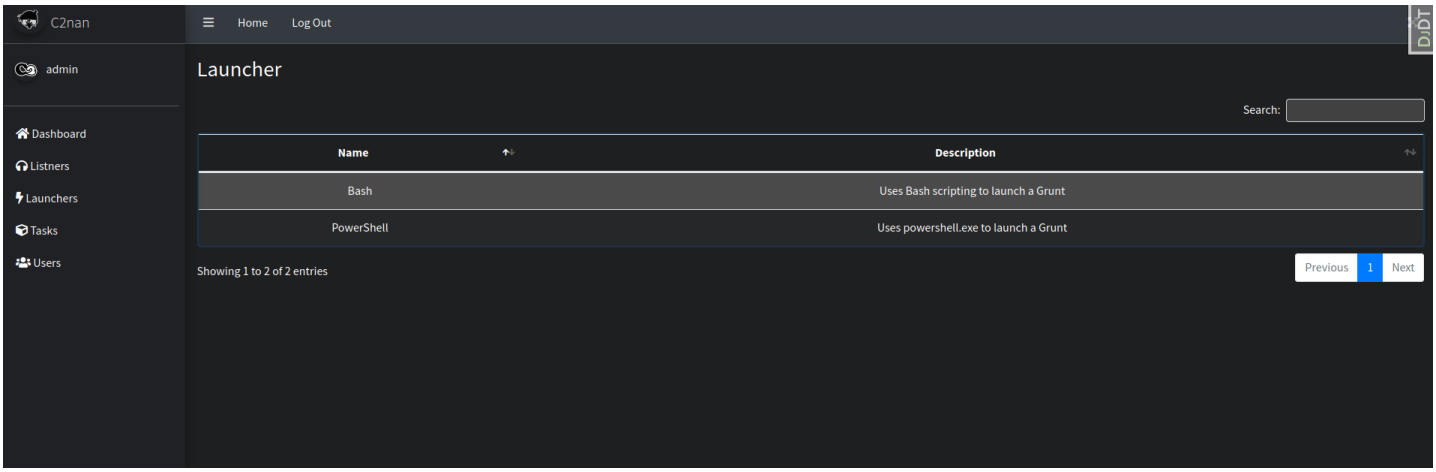

Figure 34 Launcher page

➢ Enter Un-existing Interface Case:

When selecting either bash or PowerShell on the launcher page, the GUI prompts the security engineer to enter the “network interface” and “jitter” values, mirroring the validations used in the listener creation process. Interestingly, if an invalid or non-existent network interface is entered, the tool behaves as expected by displaying a "**saved failed**" message without actually saving the configuration. In this test case, the behavior is considered a **pass**.

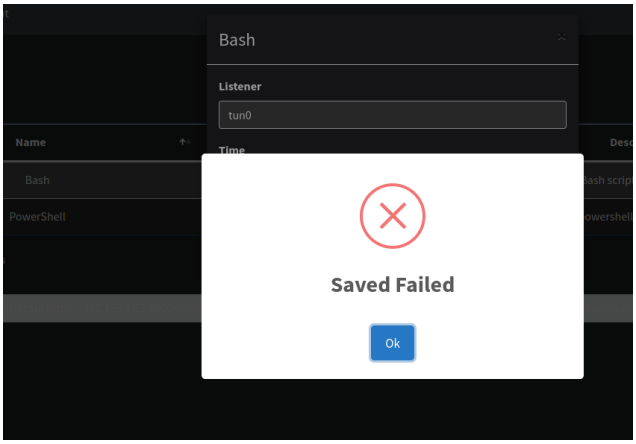

Figure 35 Launcher interface test

➢ Successful payload creation Case:

In the test case where a launcher configuration is successfully created, the corresponding payload is displayed on the home screen, just like in the listener creation process. This consistent behavior ensures that security engineers have visibility into the generated payload.

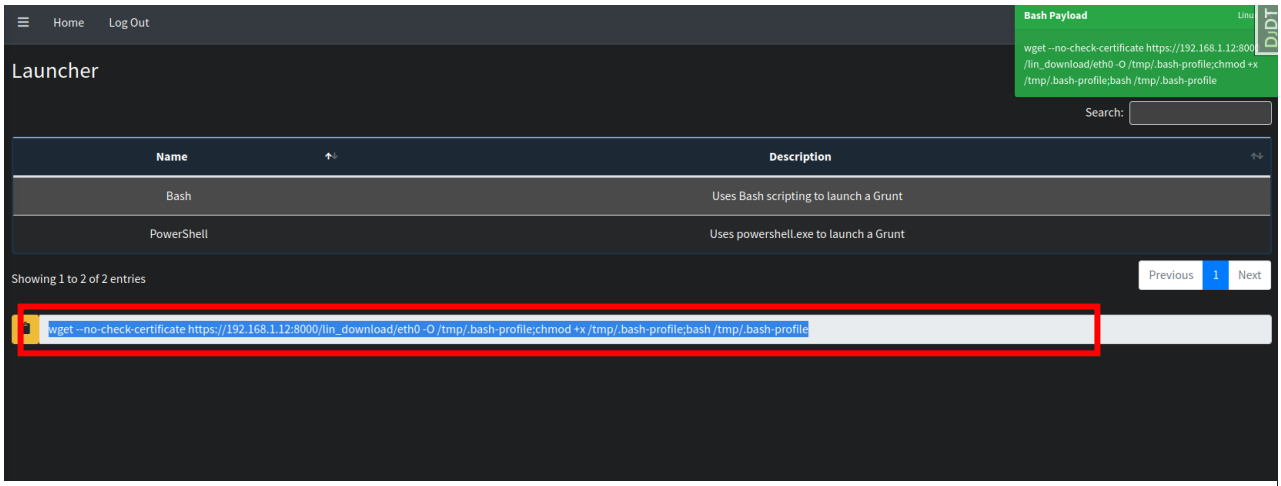

Figure 36 Launcher Payload Creation

8.3 Agent Testing

The agent section of the tool encompasses test cases that primarily revolve around the creation process, as well as the underlying functionality and logic when connecting to an agent. While these test cases may not be directly visible in the GUI, they can be monitored through the logs, which

capture the requests and responses exchanged between the server and the agent.The focus of these test cases lies in evaluating the registration process of the agent and ensuring that it successfully establishes a connection with the server. The testing will encompass various aspects of communication between the server and the agent, verifying the reliability and integrity of data exchange. Monitoring the logs allows security engineers to gain insights into the details of the requests and responses, enabling them to assess the effectiveness of the communication protocols implemented. By carefully analyzing the logs, security engineers can validate the accuracy of the exchanged data and identify any potential issues that may arise during the agent-server interaction.

- **Registration Case:**

After executing the payload generated in the preceding sections on the agent, an important test case focuses on monitoring the agent's behavior during registration and subsequent communication with the server. Upon execution, the agent initiates a registration process by sending a POST request to the server. This test case verifies the successful completion of the registration process, ensuring that the agent is properly identified and recognized by the server. Once the registration is confirmed, the agent commences regular communication by sending GET requests to the server. These GET requests contain a unique, randomly generated six-character name assigned by the server, such as **"JOOLIF."** This test case **passed successfully**.

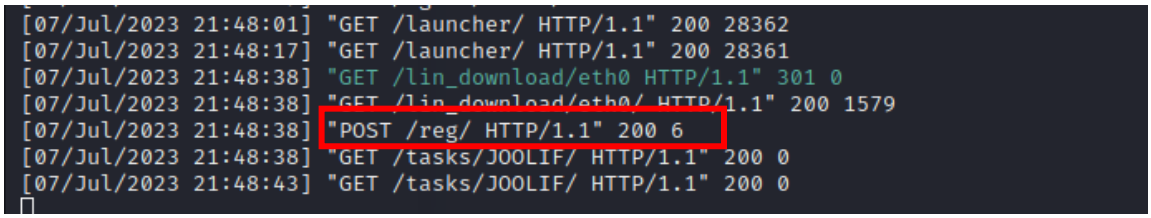

Figure 37 Agent Registration

- **Reconnection Test Case:**

In cases where the connection between the agent and server is lost for any reason, the agent is designed to **persistently** attempt to reconnect to the server. Upon successful reconnection, the agent undergoes the registration process again, allowing the server to recognize and establish communication with the agent. During the re-registration, the server assigns the agent the **same unique six-character name** that was initially provided. This ensures consistency in the agent's

identity and allows the server to associate the reconnected agent with its previous data and configurations.

```
[07/Jul/2023 21:49:04] "GET /tasks/JOOLIF/ HTTP/1.1" 200 0
[07/Jul/2023 21:49:10] "GET /tasks/JOOLIF/ HTTP/1.1" 200 0
[07/Jul/2023 21:49:15] "GET /tasks/JOOLIF/ HTTP/1.1" 200 0
[07/Jul/2023 21:51:02] "GET /lin_download/eth0 HTTP/1.1" 301 0
[07/Jul/2023 21:51:02] "GET /lin_download/eth0/ HTTP/1.1" 200 1579
[07/Jul/2023 21:51:02] "POST /reg/ HTTP/1.1" 200 6
[07/Jul/2023 21:51:02] "GET /tasks/JOOLIF/ HTTP/1.1" 200 0
```

Figure 38 Re-connection Test

Additionally, if the **agent's IP** address has **changed** since the last connection, the server updates the agent's information in the database accordingly. This ensures that the server maintains accurate and **up-to-date records** of each agent's IP address, facilitating effective communication and tracking of agent activities. The updated agent information, including the new IP address, if applicable, is reflected in the dashboard's agents' table. This allows security engineers to easily monitor and track the status and connectivity of each agent, ensuring visibility and control over the network of agents. This test case has **passed successfully** with all conditions.

- **Multi-agent connection Test Case:**

During the assignment process, the agent can establish a connection with the server using different usernames based on the agent's privilege level within the system. For instance, the agent may initially connect using a **low-privilege** username such as **"normal/hasan."** Subsequently, if the security engineer acquires **higher privileges** on the same device, they can execute the tool payload

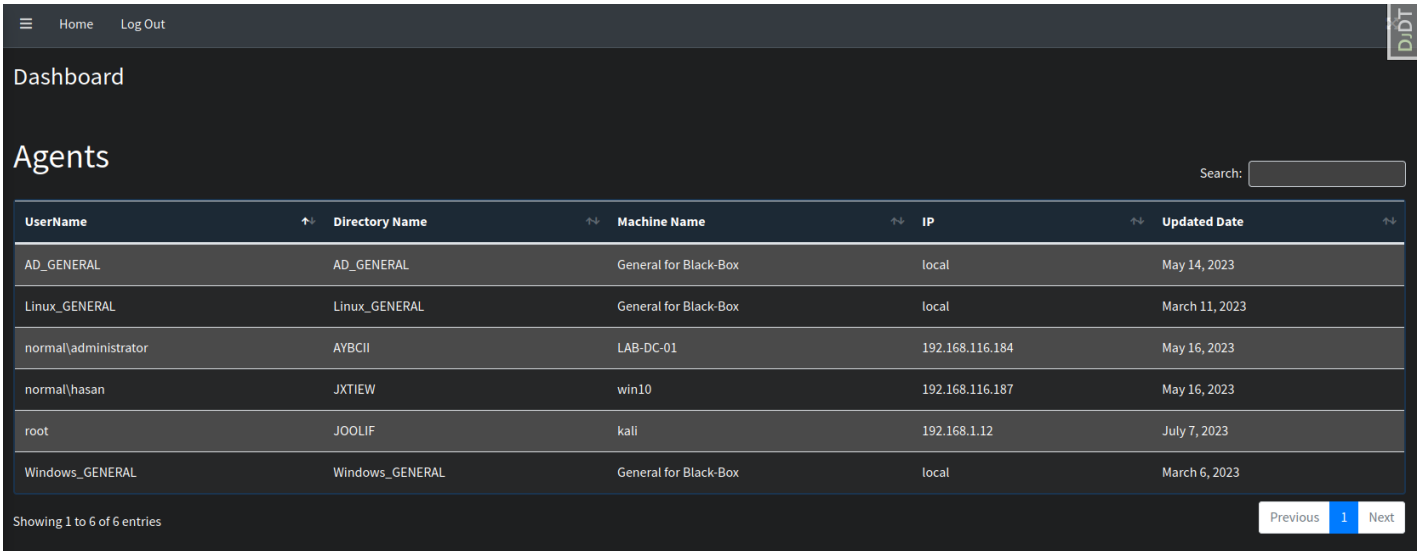

Figure 39Multi-agent connection

to establish a new connection with an elevated privilege username like **"normal/administrator."** When the security engineer connects as a new agent with the higher privilege username, the server recognizes it as a distinct agent with a unique six-character name, differentiating it from the previously connected low-privilege agent. Both agents, with their respective usernames and names, are displayed on the dashboard, providing a clear overview of their presence and capabilities. This segregation of agents based on their privilege levels allows for differentiated task assignment and execution. The security engineer can assign specific tasks to each agent based on their privileges, with the higher-privileged agent being capable of performing more privileged operations and administrative tasks. This example provides the capability of the tool for handling multi-agents, and this provides that this test case has **passed successfully.**

8.4 Tasks Testing

In the final section, the focus is on testing the tasks performed on the agent. It is important to note that **all the modules developed within the tool have undergone thorough testing and are confirmed to be working efficiently**. Therefore, this chapter does not involve testing the functionality of the modules but instead focuses on specific edge cases that may arise during task execution.

➢ Validations Test Case:

To begin, upon selecting an agent, a dedicated page is displayed. If the security engineer attempts to perform a task without selecting the operating system (OS), the tool incorporates validations to ensure proper usage. The validations are designed to detect when the OS selection is missing and prompt the engineer to choose the appropriate OS before proceeding. So, this test case has **passed successfully.**

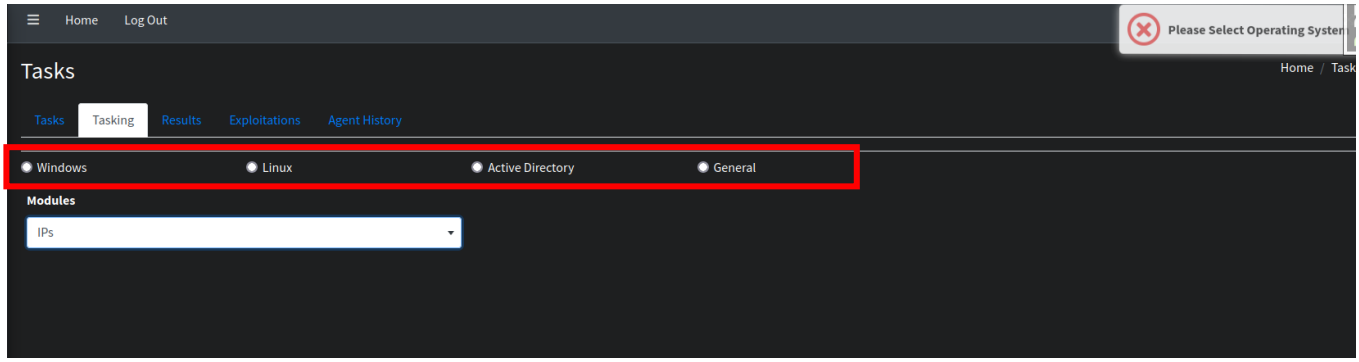

Figure 40 Tasks OS Validation

➢ Cross-Operating system task execution test case:

In the specific test case where a task intended for Windows modules is executed on a Linux operating system, an interesting observation arises. Despite the lack of warning messages, the task is executed, but the results obtained are in the form of unintelligible or garbled sentences. This unexpected outcome indicates a **failure in the test case**. While the task is executed without raising any warning or error messages, the nonsensical results indicate that the task's functionality is compromised when executed on an incompatible operating system.

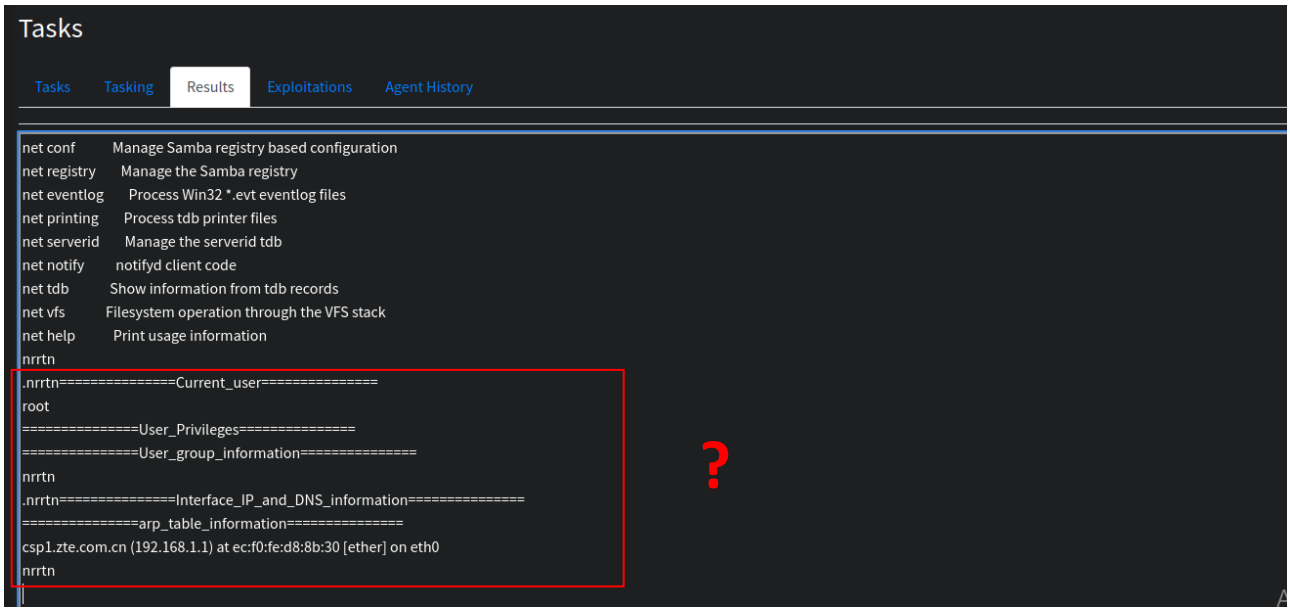

Figure 41 False results

Addressing this compatibility gap becomes crucial to enhance the overall performance and effectiveness of the tool. During the **improvement phase**, efforts will be made to resolve this issue and introduce mechanisms that prevent the execution of incompatible tasks on inappropriate operating systems. This ensures that tasks are executed only within compatible operating system environments, mitigating the risk of obtaining erroneous or misleading results.

➢ Long-Running Module Execution test case:

During the testing process, a specific test case involving the execution of a module that requires a substantial amount of time or produces lengthy results was conducted. The objective was to evaluate the tool's capability to handle such scenarios without encountering any issues. This **test case was successfully passed**.

When running the module, which involved a lengthy execution or produced long result outputs, the tool demonstrated its robustness by executing the task without any failures or crashes. The channel used for communication between the agent and the server remained stable and uninterrupted throughout the entire execution process.

The successful execution of this test case demonstrates the tool's ability to handle resource-intensive tasks effectively, ensuring that tasks requiring extensive execution time or generating large result outputs can be completed without any disruptions. The tool's stability in handling long-running modules is crucial for maintaining reliable and uninterrupted communication, allowing security engineers to efficiently analyze the results and make informed decisions based on the outcome.

➢ Module’s input validations enhancement:

During the testing phase, it was observed that the input of the modules **lacked comprehensive validations**. This is due to the nature of the inputs, as it is not always feasible or desirable to enforce strict constraints on the security engineer's input.

In the **improvement phase**, the focus will be on developing an efficient and effective solution for input validation in modules. This involves finding a balance between providing flexibility to security engineers while also ensuring that inputs conform to expected formats and meet essential criteria.

Chapter 9
System Results & Scenarios

Chapter 9: System Results and Scenarios

9.1 Linux Complete Scenario

In this particular scenario, we were given access to the network but had no initial access to any agent. To detect the targeted agent, we began by scanning **the network** by discovering the ARP packets to show all connected IPs and their corresponding MAC addresses. After analyzing the results, we targeted a specific IP address, such as **192.168.116.136**.

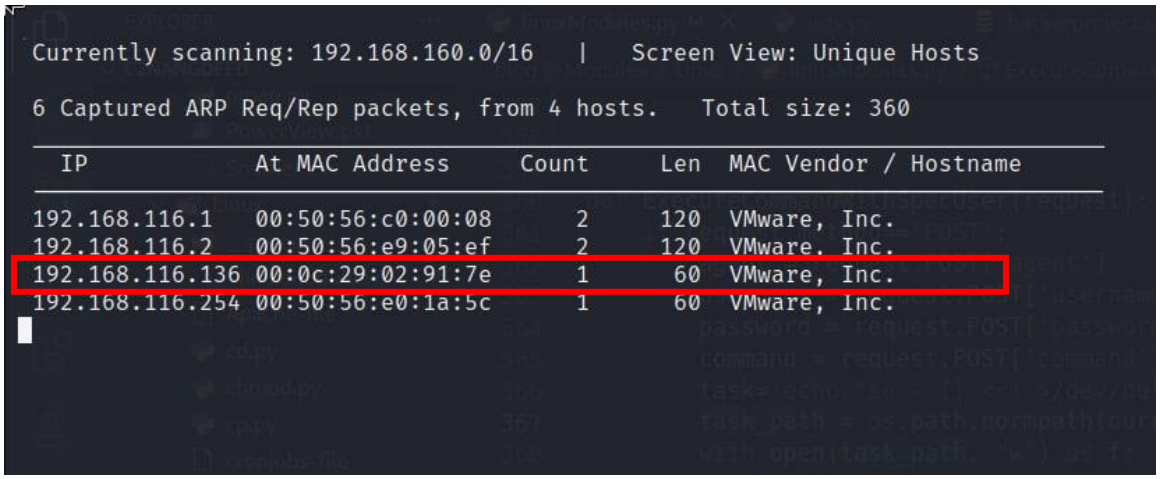

Figure 42 Netdiscover results

To perform the **enumeration** on the agent with IP address **192.168.116.136**, we scanned all opened local ports. Our findings showed that **the HTTP port (port 80) was open**, indicating the presence of a web server. We accessed the web server by entering **http://192.168.116.136/** into our browser's URL bar, and we made sure of that point.

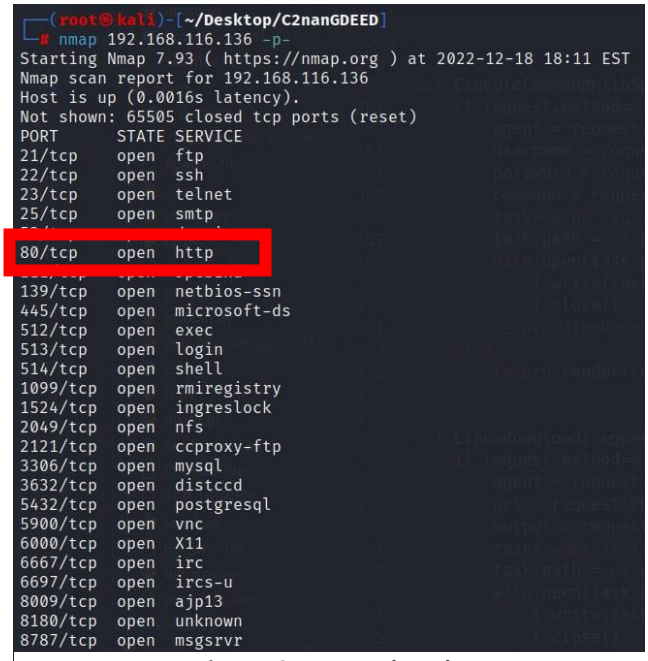

Figure 43 nmap tool results

Our C2 framework now comes into play as we begin our next step: fuzzing the URL. This is done using a built-in module in our framework that employs a technique called **URL fuzzing**. Its purpose is to uncover any hidden files or directories in the web server, which may contain sensitive information such as a database or web server backup files, log files, testing pages, and more. By utilizing this method, we can identify potential vulnerabilities and exploit them to gain access to the system.

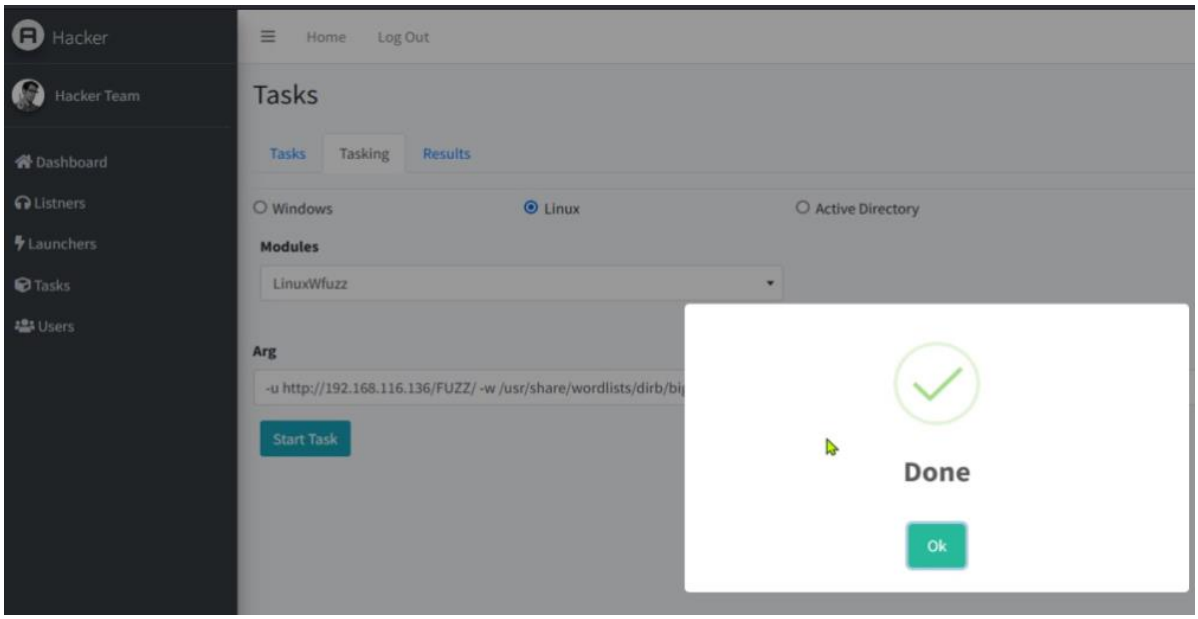

Figure 44 URL Fuzzing

During our URL fuzzing process, we discovered a directory named **/dav**. This directory is associated with **WebDAV**, which stands for **"Web-based Distributed Authoring and Versioning."** It's a set of HTTP protocol extensions that enable users to manage and edit files on remote web servers. WebDAV can be a useful tool for collaborative editing and version control, but it can also present a security risk if not configured correctly. Therefore, we decided to investigate further and determine if this WebDAV implementation on the target server is vulnerable to exploitation or not.

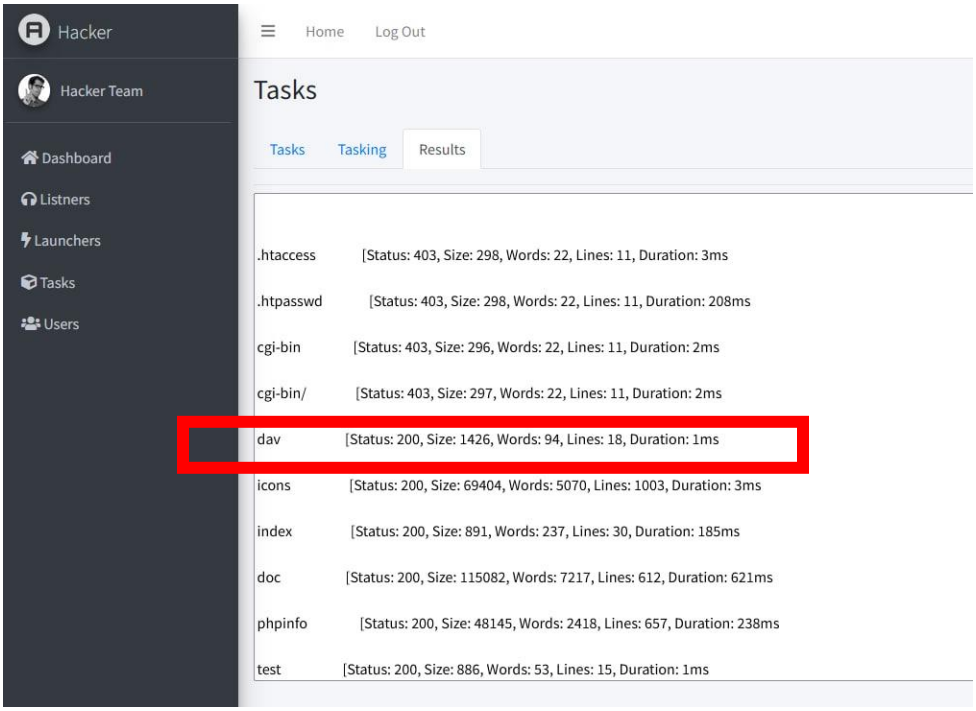

Figure 45 URL Fuzzing Results

To exploit the discovered WebDAV file, we conducted a vulnerability assessment. The results revealed that the server was vulnerable to arbitrary file upload attacks, allowing us to put **PHP scripts** on the remote web server. With this access, we could execute malicious code on the target machine and potentially gain further access to the system.

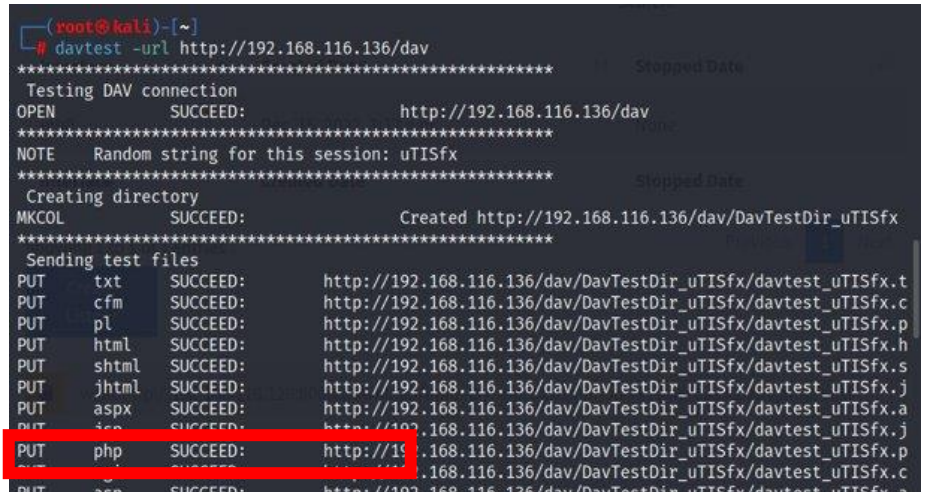

Figure 46 Testing Webdav

After discovering that we could put a PHP script on the web server, we generated a **Bash payload** using our framework to be inserted into a PHP file. The Bash payload was crafted to establish a

reverse shell connection to our C2 server. This would give us full access to the target system and allow us to execute commands remotely.

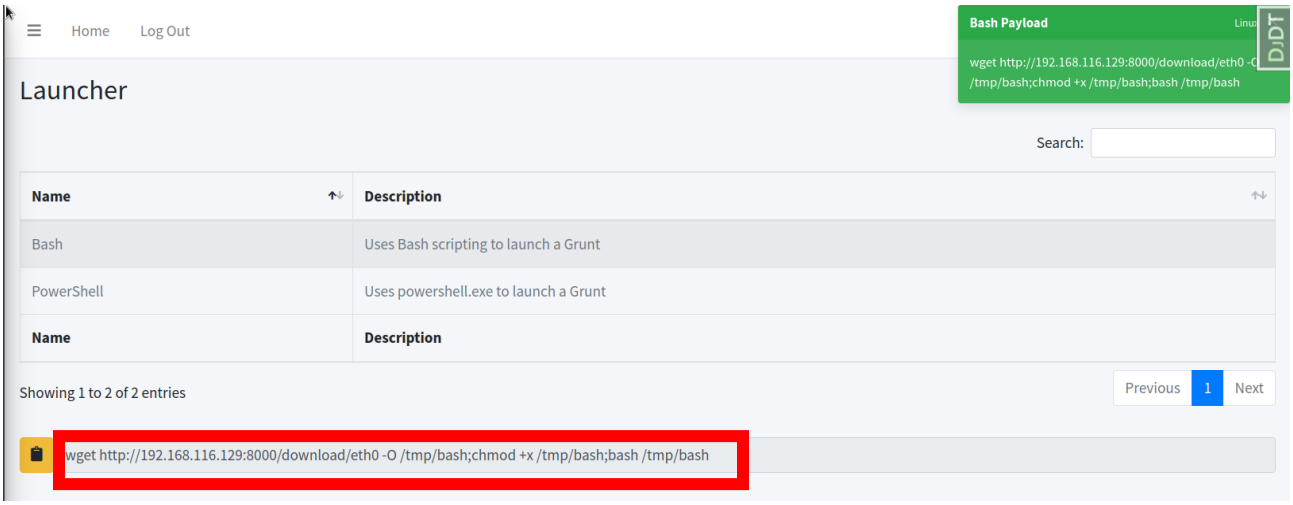

Figure 47 Generating Bash payload

The generated payload was then inserted into a PHP file and uploaded to the server through the /dav endpoint.

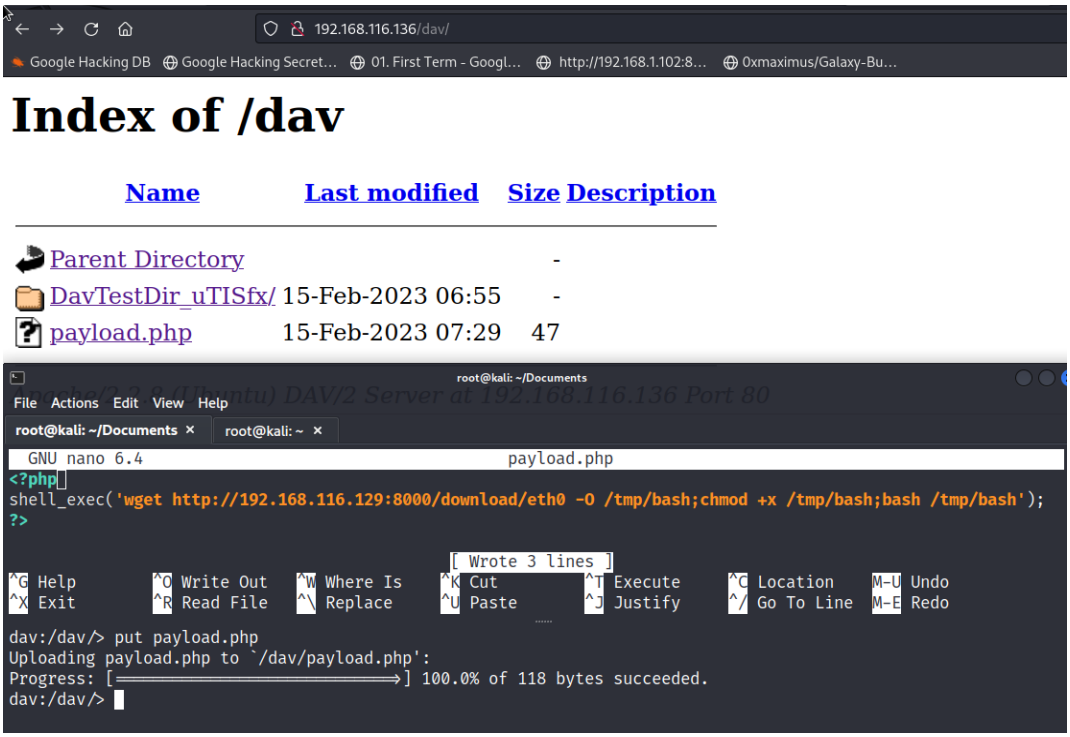

Figure 48 Inserting payload into php file

Once the file was uploaded, we executed the PHP script using a web browser and successfully established a reverse shell connection to the target system with a **low privileged user (service privilege) www-data**.

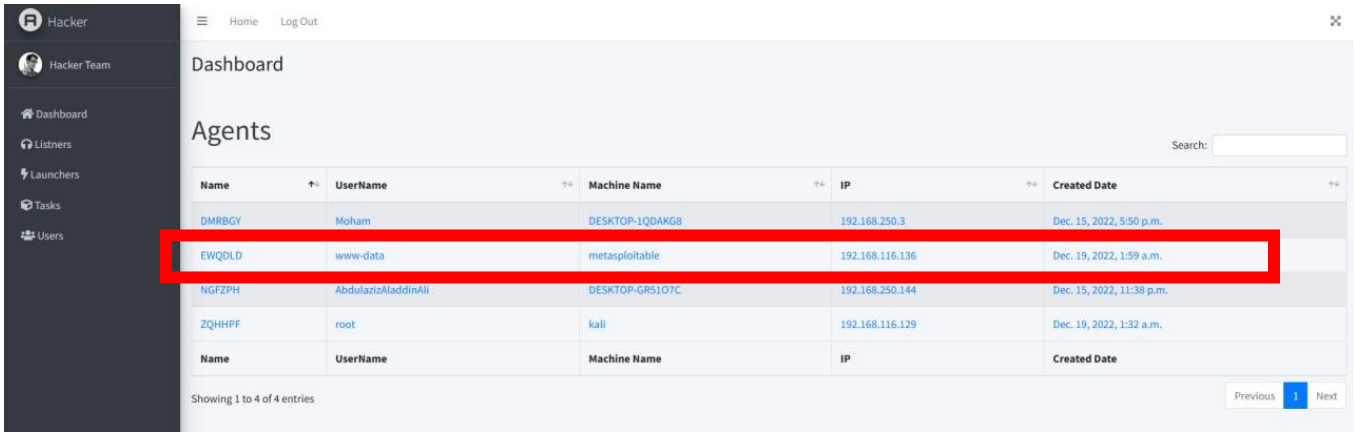

Figure 49 Exploiting WebDAV

We started running our local enumeration modules on it to gather more information about the system. These modules cover a wide range of areas, such as system configuration, installed software, system groups, user accounts, and user privileges.

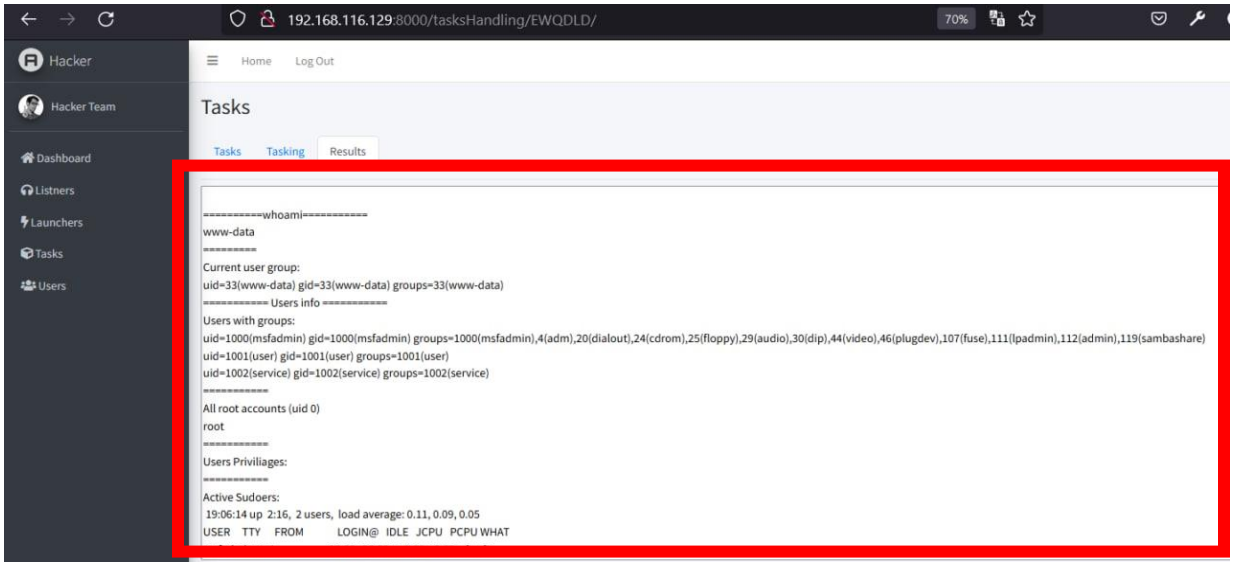

Figure 50 Performing tasks on agent

During the local enumeration of the targeted agent, we extracted the contents of the /etc/passwd file, which revealed information about all the users on the system. Interestingly, we found a user account named "user" with a comment next to it stating [Just a user].

This could indicate that the user was created for testing purposes and was not intended to be a regular account. Out of curiosity, we attempted to log in with the username "user" and the password "user," and to our surprise, it worked. This discovery could potentially provide us with additional access and privileges on the agent.

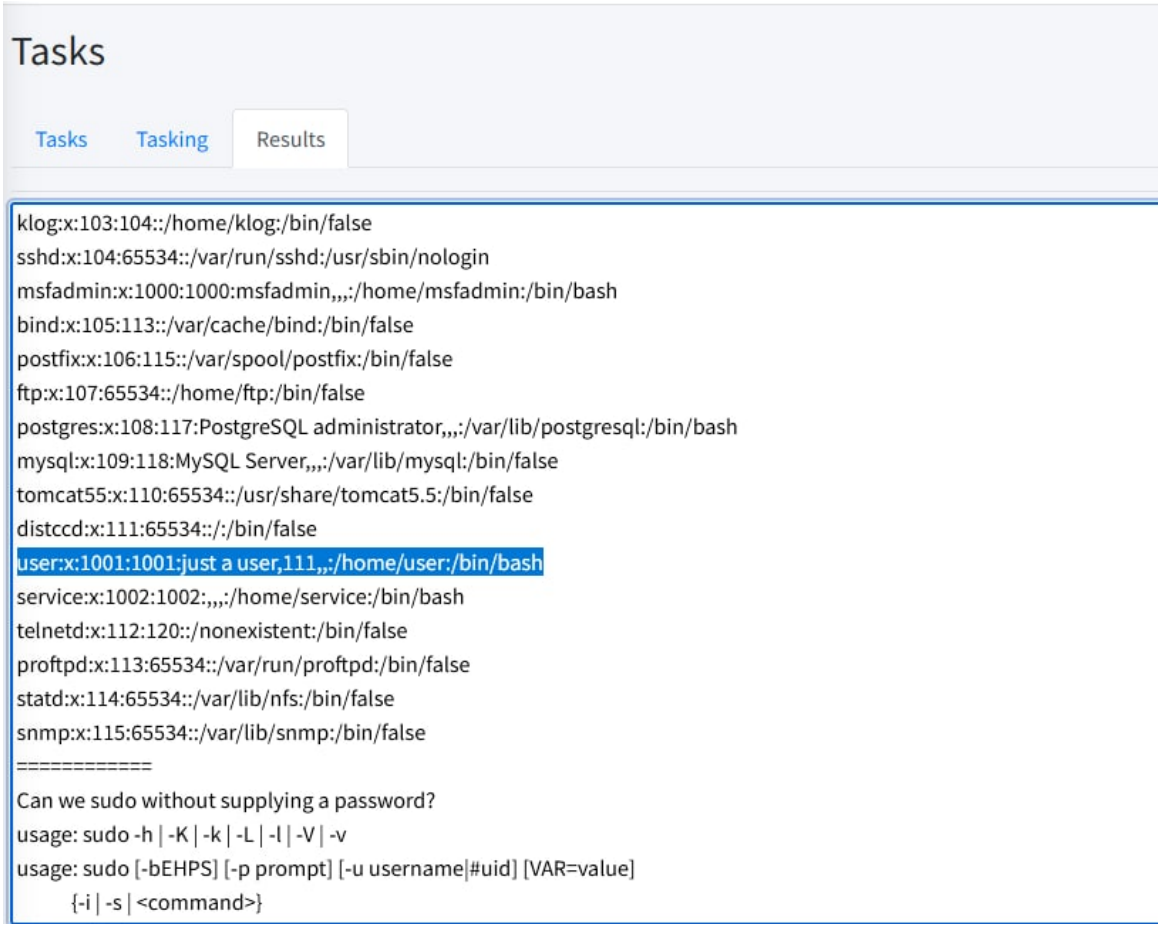

Figure 51 cat /etc/passwd

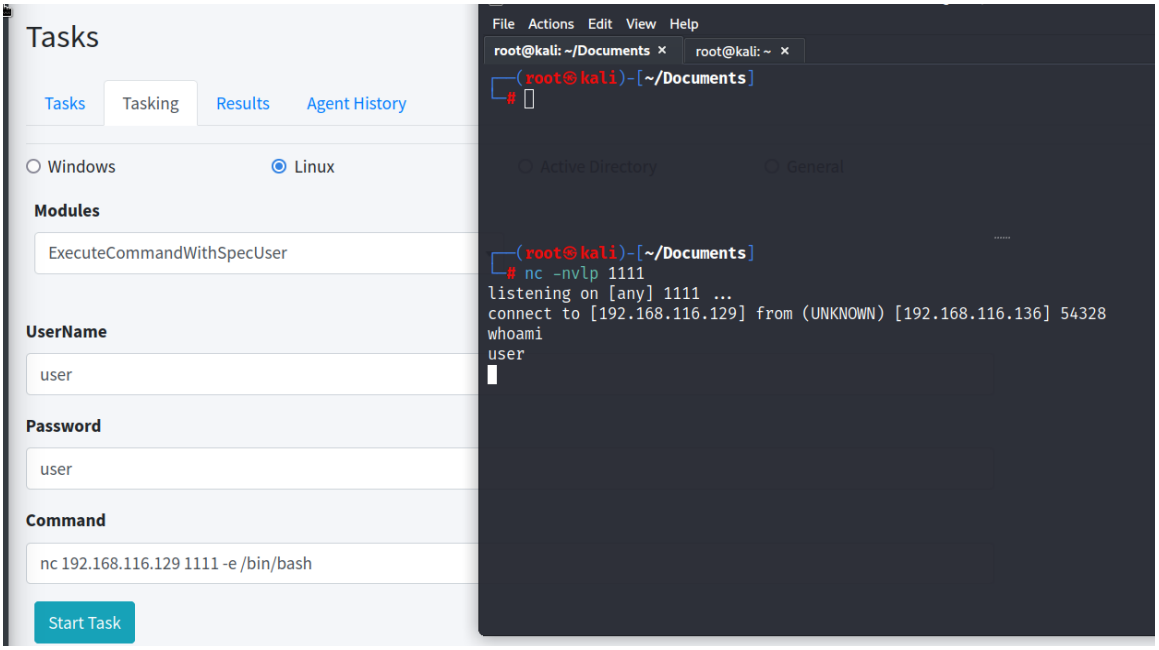

Figure 52 Execute commands with a user credential.

After discovering a valid user account named "user" on the target system, we used our framework's modules to execute a reverse shell with the privileges of this user. This allowed us to gain access to the system's command line interface and execute commands as the user "user."

With the reverse shell established, we were able to gather additional information about the system, such as its operating system version, running processes, and installed software. This information proved useful in identifying potential vulnerabilities and further exploiting the system. However, we always made sure to exercise caution and take appropriate measures to avoid causing any damage to the system or its data.

During our enumeration, we discovered that the operating system on the target agent was running an outdated version. After conducting further research, we found that the version was vulnerable to a known CVE (Common Vulnerabilities and Exposures). To confirm our suspicions, we cross-checked the version with the Exploit Database and found a corresponding exploit that could potentially allow us to gain unauthorized access to the system.

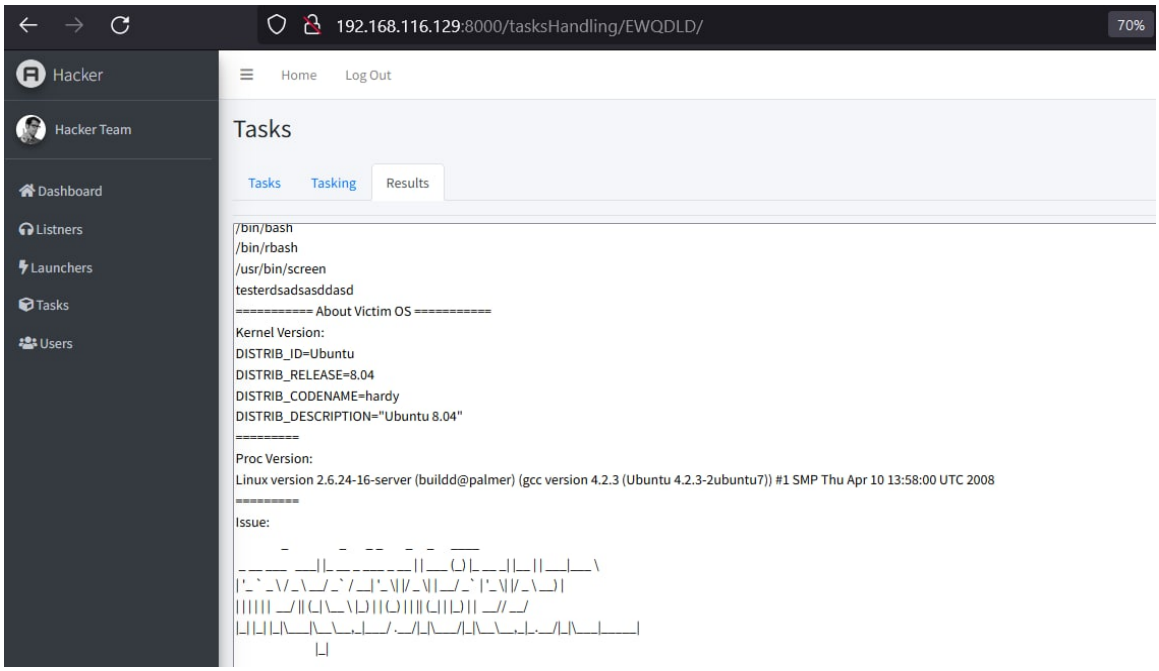

Figure 53 Get user privilege

After downloading the CVE file [dirty.c] on the Agent, by using our modules, we realized that it needed to be compiled before it could be executed through the terminal.

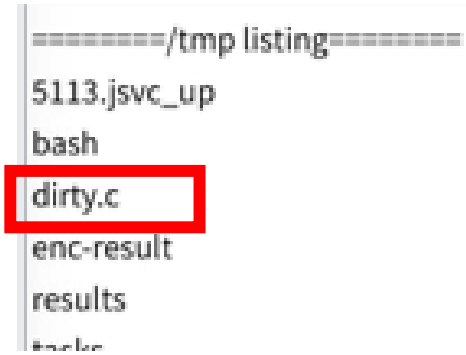

Figure 54 Info about user OS

To compile the file, we used the gcc module on our framework. This created an executable file called 'dirty' that could be run with elevated privileges.

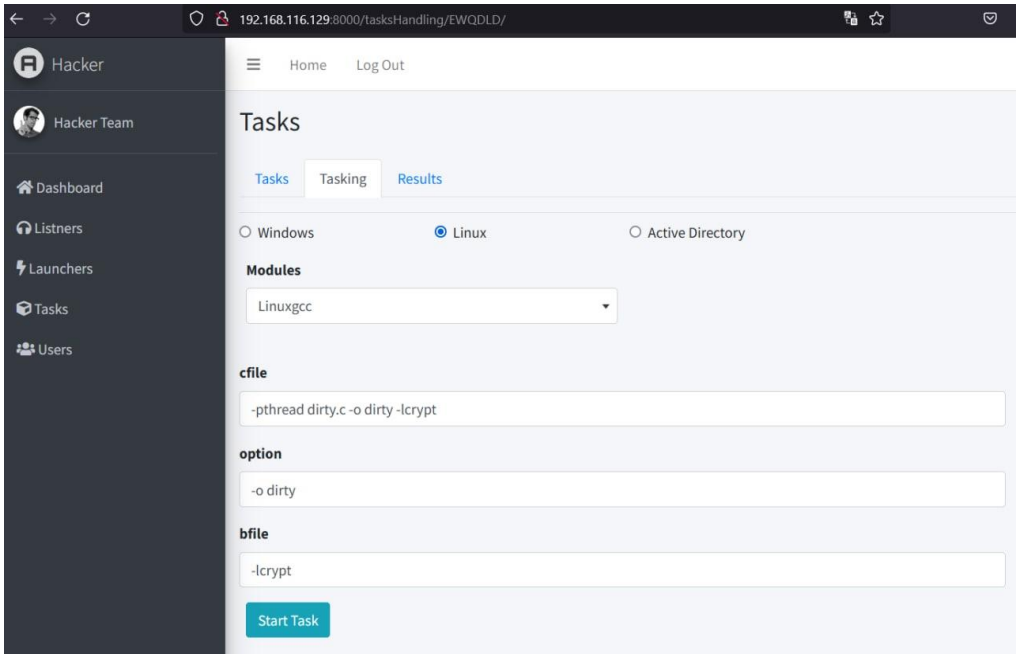

Figure 55 compiling the .c file

Once we had the executable file, we used our bash module to execute it on the agent. This resulted in escalating the user privilege to **"root",** giving us full control over the system.

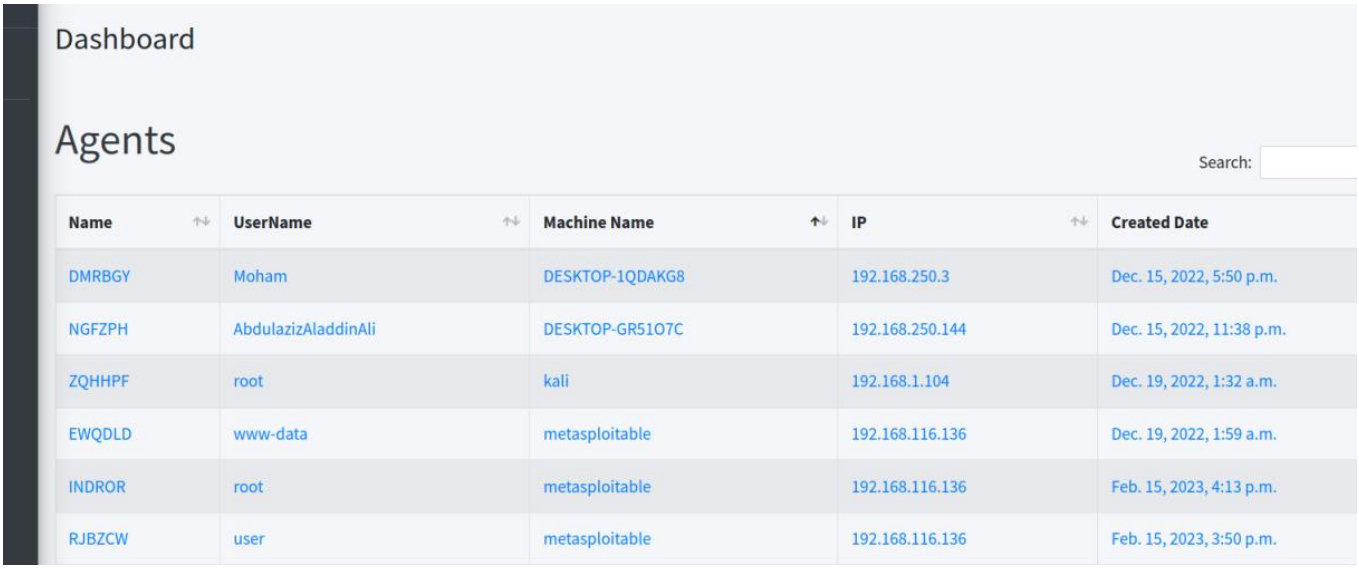

Figure 56 Executing CVE

9.2 Windows Scenario: based on privilege Escalation and Exploitation

This section discusses the concept of privilege escalation and how hackers can exploit vulnerabilities to gain higher privileges in Windows systems. We will explore a specific example of privilege escalation using the SeDebugPrivilege as a means to attain the Authority System privilege. Understanding these vulnerabilities is crucial for system administrators and security professionals to effectively protect their Windows environments.

9.2.1 Privilege Escalation

Privilege escalation refers to the act of gaining higher levels of access or privileges beyond what is initially assigned to a user or process. It allows an attacker to bypass security measures and gain unauthorized control over a system. Privilege escalation can be achieved through various methods, including exploiting vulnerabilities in software, misconfigurations, or abusing system privileges.

9.2.2 What is SeDebugPrivilege

In Windows operating systems, SeDebugPrivilege is a security privilege that allows a user or process to obtain debug privileges. Debug privileges provide extensive access to system resources and the ability to manipulate and monitor processes, threads, and system objects. This privilege is typically granted to administrators and certain system processes.

Debug privileges enable advanced debugging capabilities, including attaching a debugger to running processes, viewing and modifying the memory of other processes, and manipulating system components. These privileges are essential for certain types of software development, system troubleshooting, and security analysis tasks.

To understand SeDebugPrivilege better, here are some key points:

Privileges in Windows: Privileges in Windows define specific rights and permissions that a user or process can have. These privileges control access to various system resources and capabilities.

SeDebugPrivilege: SeDebugPrivilege, also known as the "Debug Programs" privilege, allows a user or process to debug other processes on the system. It is one of the many privileges available in Windows.

User Rights Assignment: SeDebugPrivilege can be assigned to user accounts or user groups using the "User Rights Assignment" feature in the Local Security Policy or Group Policy settings. By default, this privilege is granted to administrators.

Debugging Scenarios: SeDebugPrivilege is particularly useful in scenarios such as:

Software Development: Debuggers require this privilege to attach to and analyze running processes during software development and debugging.

System Troubleshooting: Debug privileges can help diagnose and troubleshoot issues by allowing an in-depth examination of system processes and resources.

Security Analysis: Security professionals may utilize debug privileges to analyze and investigate potential security vulnerabilities and malware.

Security Implications: Since debug privileges provide extensive access to system resources, they can be abused by malicious actors if granted to unauthorized users or processes. It is crucial to carefully manage and restrict the assignment of SeDebugPrivilege to maintain system security.

It's important to note that obtaining and using SeDebugPrivilege requires appropriate administrative rights or authorization. It is not recommended to grant this privilege to ordinary user accounts unless required for specific tasks.

9.2.3 How to manipulateSeDebugPrivillage

One of the SeDebugPrivillage abilities is to spawn a sub-process from a parent process with the same privileges as the parent process, and by this ability, we could start searching for a parent process that we know for sure that it’s running by the system, for example, winlogon.exe.

9.2.4 What is winlogon.exe

Winlogon.exe is an important system process in the Windows operating system. It is responsible for managing the user's login and logout procedures, as well as the secure attention sequence (SAS) handling. Winlogon.exe is an essential component of the Windows logon process and ensures the integrity and security of the user's session.

Some key points about winlogon.exe

Login and Logout Management: When a user logs into a Windows system, winlogon.exe is responsible for verifying the user's credentials, loading the user profile, and initializing the user's session. It manages the process of starting up the user environment, including launching the user shell (typically explorer.exe) and other startup applications. When a user logs out or shuts down the system, winlogon.exe handles the necessary cleanup and termination of processes.

Secure Attention Sequence (SAS) Handling: Winlogon.exe is responsible for processing the Secure Attention Sequence, which is a key combination (Ctrl+Alt+Delete) that allows users to securely log in, lock the computer, change passwords, and access the Task Manager. It ensures that the SAS is only processed by a trusted process (logonui.exe) and helps prevent spoofing or interception by malicious software.

System Integrity: Winlogon.exe operates in the context of the Local Security Authority (LSA) and is considered a trusted process. It runs with high privileges to perform critical system operations, such as loading authentication providers, managing security policies, and enforcing user rights and privileges. Its integrity and security are essential for maintaining the stability and protection of the operating system.

Location and File Details: Winlogon.exe is typically located in the C:\Windows\System32 folder. It is a Windows system file and should not be modified or deleted. It has a digital signature from Microsoft to ensure its authenticity and integrity.

Malware Impersonation: Since winlogon.exe is a critical system process, some malware may attempt to impersonate it by using a similar name or location. This is done to deceive users and gain elevated privileges. It is important to regularly scan your system for malware and ensure that the winlogon.exe process is legitimate by verifying its digital signature and file location.

In summary, winlogon.exe is a vital component of the Windows logon process, responsible for user authentication, session initialization, and SAS handling. It plays a crucial role in maintaining the security and integrity of the operating system.

Now, let's initiate our scenario after comprehending the concepts of SeDebug and Winlogon.

9.2.5 Windows Scenario Procedures

1. To verify the status of SeDebugPrivilege, run the user_info module in the C2nan framework, which will provide information regarding its enabled or disabled state.

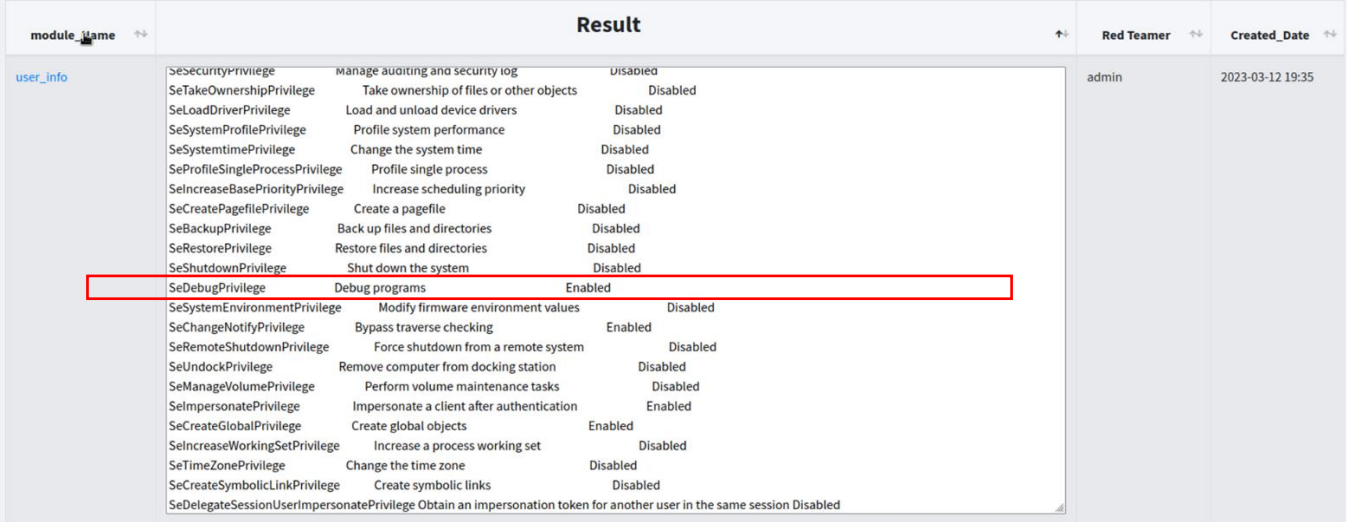

Figure 57 User_Info module results

Confirmed! SeDebug Privilege is enabled, as indicated as shown in Figure 57.

2. To find the Winlogon.exe process and retrieve its Process ID (PID), check the task list for the Winlogon.exe process and obtain the corresponding PID.

- Referring to Figure 58, the PID of winlogon.exe is indeed **556**.

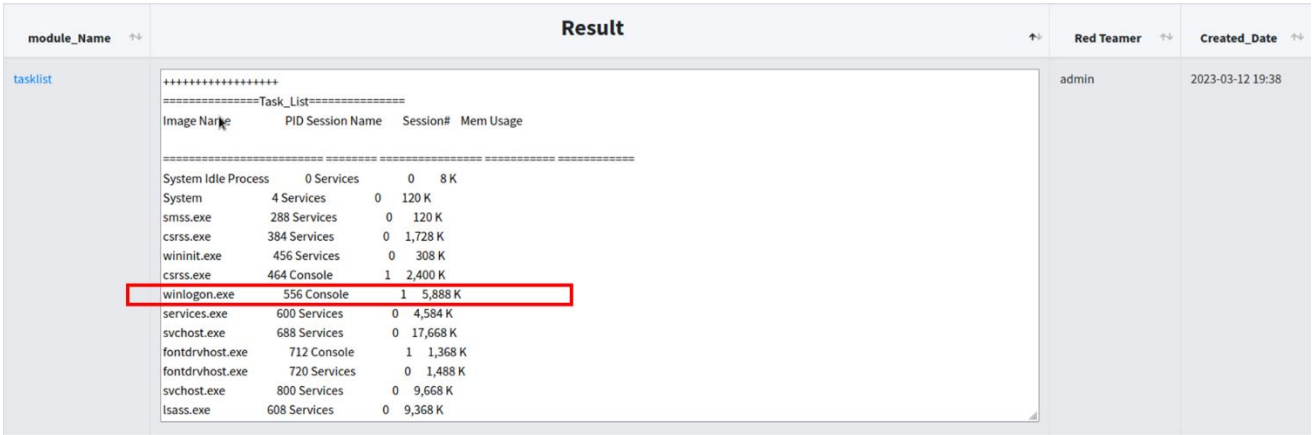

Figure 58 Checking Winlogon.exe PID

3. Next, proceed to utilize the SeDebug Exploitation module within the C2nan framework, as shown in Figure 54.

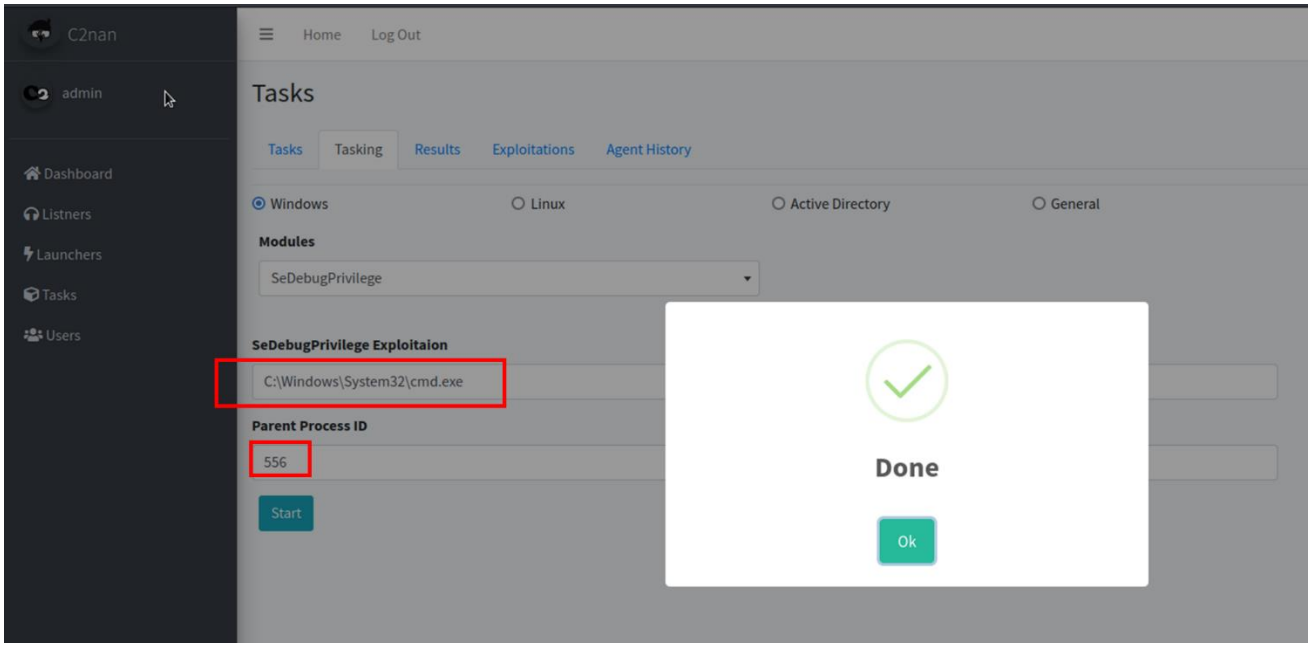

Figure 59 SeDebug Privilege Exploitation

4. By executing the module depicted in Figure 59, we successfully spawned a cmd.exe or commanded prompt process, inheriting the parent process as Winlogon with NT Authority System privilege. This was achieved by providing the C2 with the PID of the Winlogon process and the path to the cmd.exe executable.

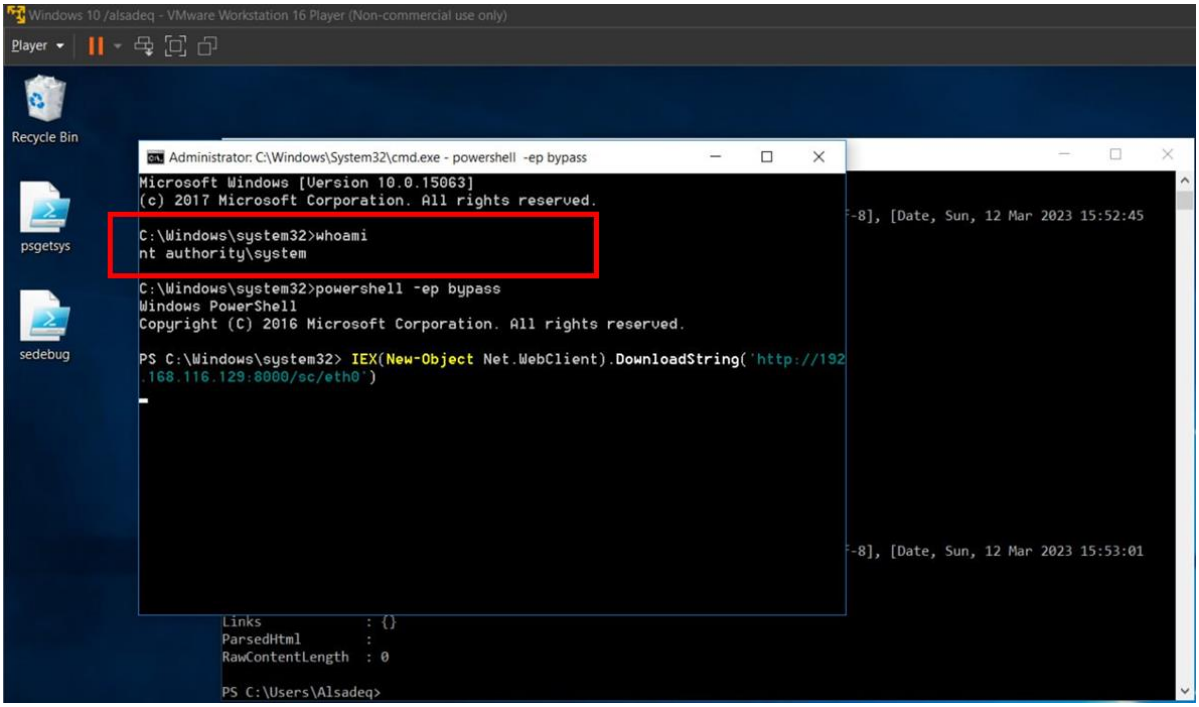

Figure 60 Exploitation POC

To leverage the newly acquired NT Authority System privilege, we can utilize the Payload Generation feature to create a payload that will establish a persistent session with our C2. This session will be recorded and accessible in our Dashboard, granting us continued access with the elevated privileges of the NT Authority System, as shown in Figure 61.

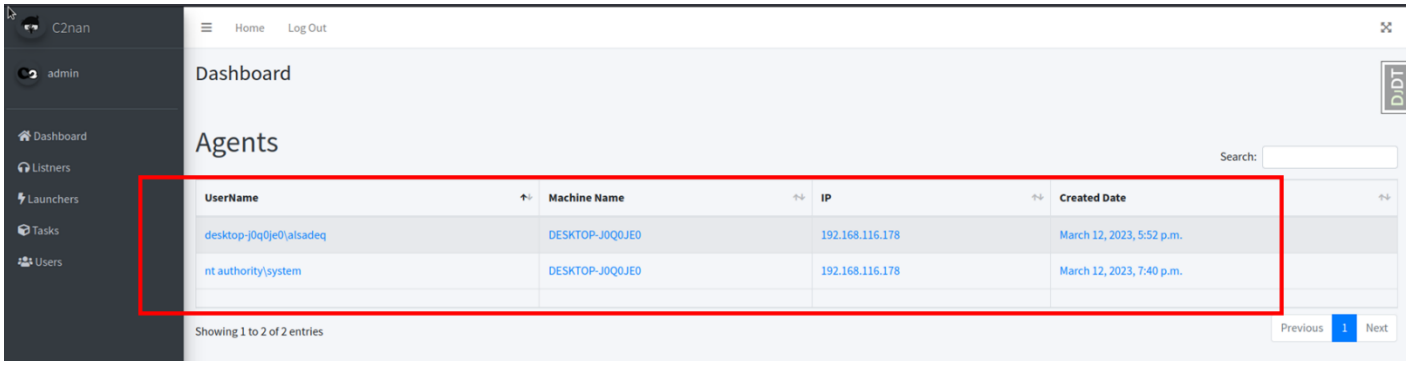

Figure 61 NT Authority agent

- In the provided scenario, the entire process **can be automated** without the need for manual inputs, making it more efficient and streamlined. By implementing PowerShell commands and regex filtering techniques, the PID of the Winlogon process can be determined automatically. Once theSeDebugPrivilege is confirmed to be enabled, the Exploitation module can be executed, resulting in the automatic spawning of a Command Prompt without the need for additional user inputs.

This proof of concept (POC) showcases the **automation capabilities integrated into the solution**, demonstrating the seamless flow of the privilege escalation process. With this automation, the entire procedure can be executed swiftly and effortlessly, enhancing efficiency and convenience for security professionals and system administrators.

As an additional feature, the SeDebug Auto Exploitation module shown in Figure 62 allows the security engineer to execute any desired file using the NT Authority System privilege automatically. This capability empowers the engineer to run various commands or scripts seamlessly, leveraging the elevated privileges for performing specific tasks or investigations. By **manualizing** the input process for just the file path, the engineer gains the flexibility to execute custom files while benefiting from the NT Authority System privilege, further enhancing their capabilities in system analysis and security operations.

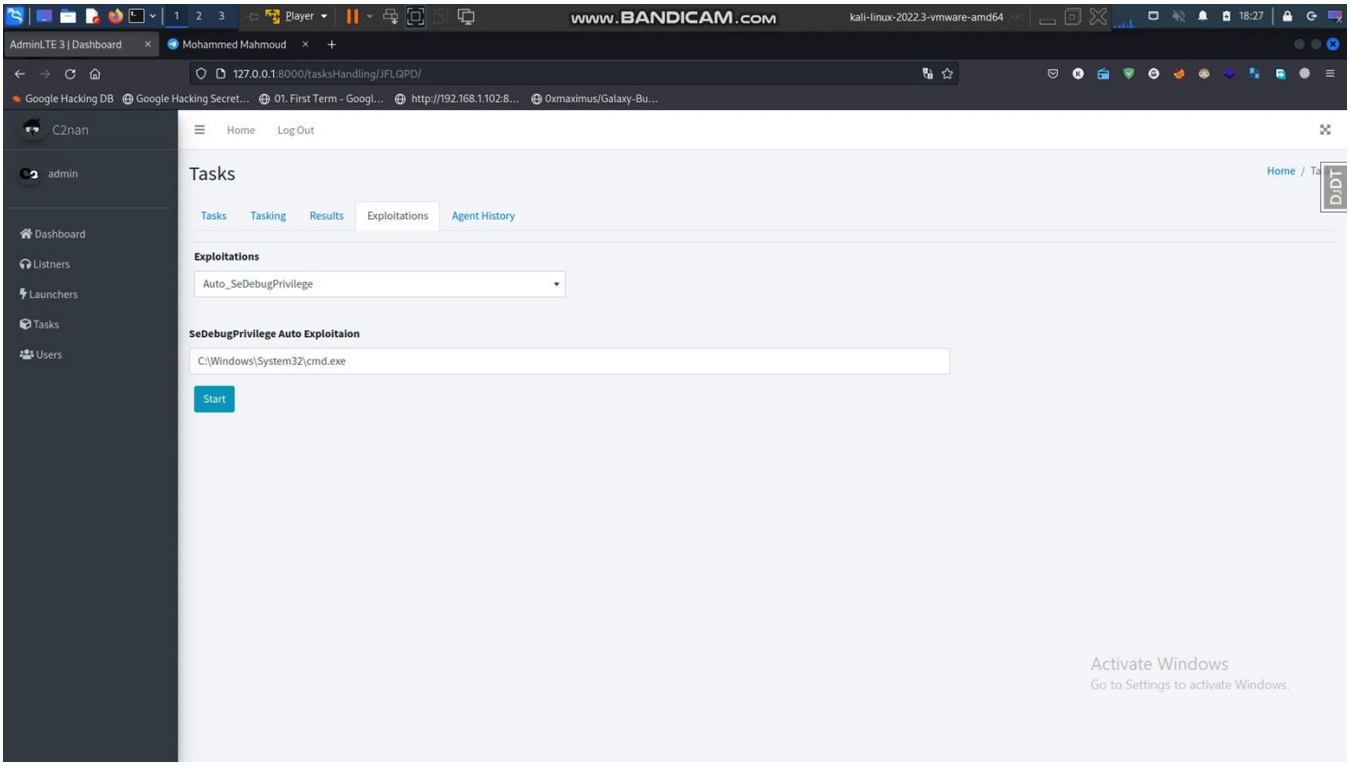

Figure 62 Automated SeDebug Exploitation

9.3 AD Complete Scenario

9.3.1 Introduction

This scenario explores the exploitation of an Active Directory (AD) environment using the C2nan C2 framework. It begins with a machine that is not joined to the AD domain, simulating an external entity attempting to gain unauthorized access. The objective is to demonstrate the potential vulnerabilities within AD and showcase the effectiveness of C2nan in escalating privileges to ultimately reach the domain administrator level.

By leveraging C2nan as a powerful command-and-control framework, various attack techniques are employed to exploit weaknesses within the AD infrastructure. The scenario follows a step-by-step process, starting with reconnaissance and enumeration to gather information about the AD

network. Subsequently, the attacker exploits identified vulnerabilities, such as weak user credentials or misconfigured access controls, to gain initial access to the AD environment.

Once inside the AD network, the attacker utilizes C2nan's capabilities to establish persistence, evade detection mechanisms, and escalate privileges. Through lateral movement and privilege escalation techniques, the attacker navigates through the AD hierarchy, targeting higher-level user accountsand system privileges. The ultimate goal is to compromise the domain administrator account, granting full control over the AD environment.

Throughout the scenario, various attack vectors and tactics are employed, including social engineering, phishing, and the exploitation of known vulnerabilities in AD components. The documentation aims to highlight the risks associated with these attack vectors and emphasize the importance of implementing robust security measures to protect AD environments.

By simulating this comprehensive scenario, the project intends to raise awareness about the potential threats and provide insights into mitigating strategies to safeguard AD infrastructures. It showcases the significance of utilizing frameworks like C2nan to understand and counteract the techniques employed by adversaries targeting Active Directory systems.

9.3.2 Scenario Procedures

The scenario begins with a non-joined machine in the AD environment, prompting an exploration of attacking the AD using a black-box approach. (Figure 63).

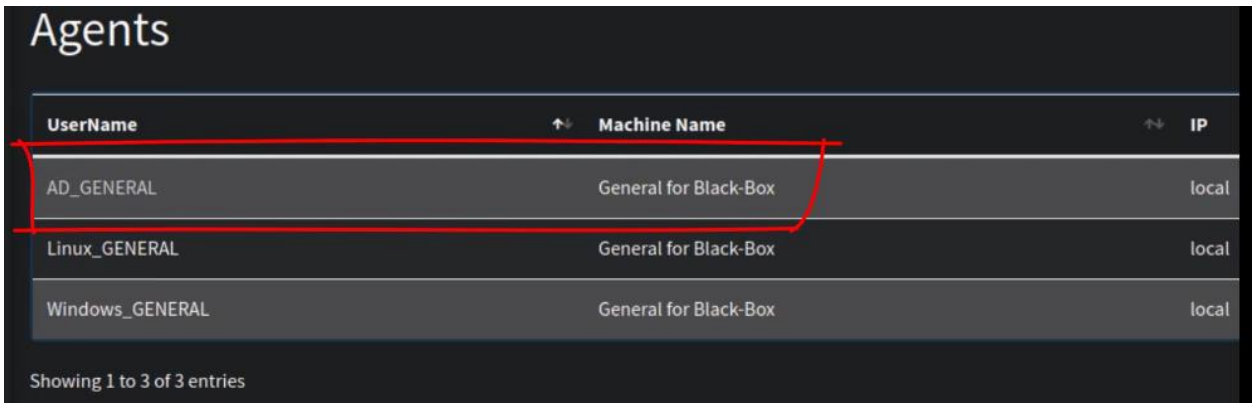

Figure 63 Blackbox attacking.

In this black-box scenario, the absence of any AD domain-related usernames and passwords leads us to employ brute-force enumeration techniques using our UserEnumKerbrute module (Figure 64). This module enables the systematic enumeration of valid AD users, enhancing our understanding of the AD environment and facilitating the subsequent stages of the attack (Figure 65).

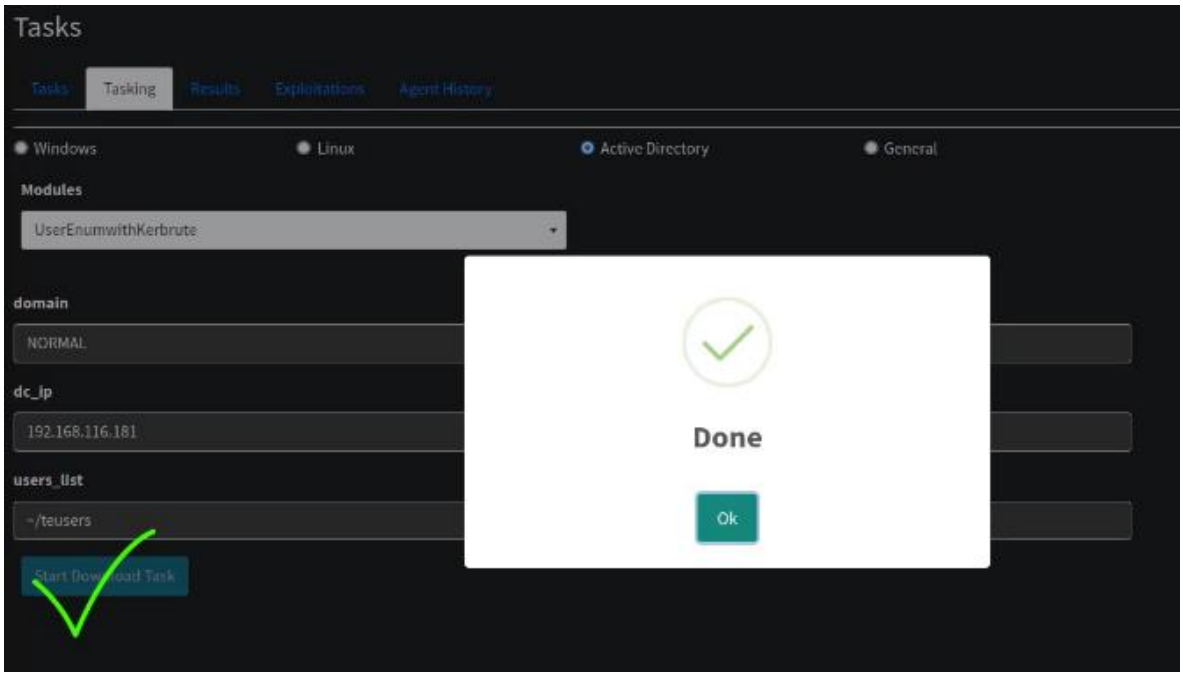

Figure 64 Enumerating valid users.

Figure 65 Valid users.

Having obtained the list of valid users, our next step in this black-box scenario is to conduct password spraying using the widely recognized and frequently utilized complex password:

P@ssw0rd (Figure 66). This password spraying technique allows us to systematically test a common credential against multiple user accounts, aiming to identify any instances of weak or compromised passwords within the AD environment.

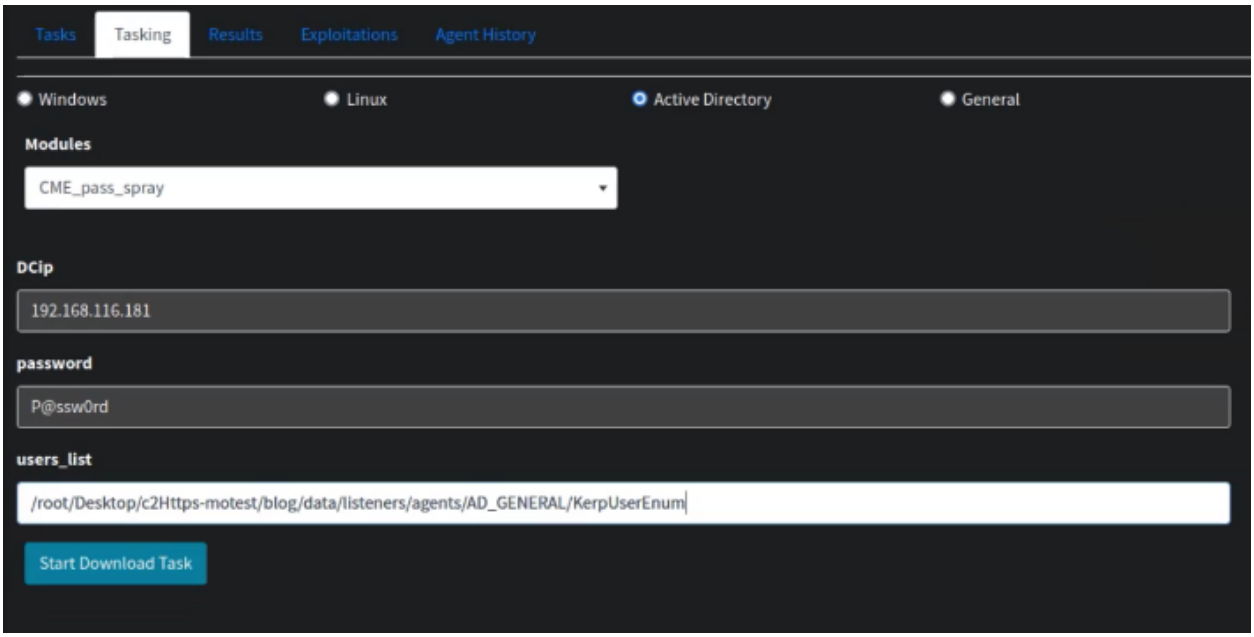

Figure 66 Password Spraying

As depicted, our efforts in password spraying have resulted in the successful capture of a user named "Ahmed" utilizing the complex password (Figure 66). This discovery highlights the importance of employing strong and unique passwords within the AD environment, as compromised credentials can provide adversaries with unauthorized access to sensitive resources Figure 67.

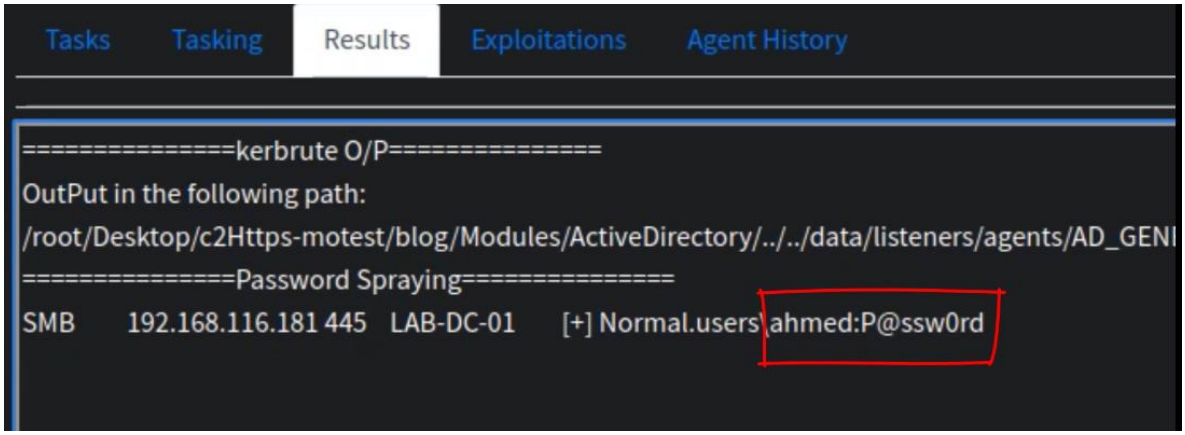

Figure 67 Password Spraying output.

Leveraging the obtained credentials, we proceed to examine the shared files within the SYSVOL directory, as it is common for regular AD users to have access to these files (Figure 68). This investigation aims to identify any sensitive information or potential vulnerabilities that may be present within the shared files, highlighting the importance of properly securing and restricting access to such resources within the AD environment.

Figure 68 Logging as smbclient.

Upon conducting an enumeration of the shares in SYSVOL, our attention was drawn to the discovery of the Groups.xml file (Figure 69). Subsequently, we transferred this file to our operating system and proceeded to open it to examine the provided credentials. However, we encountered a challenge as the passwords were encrypted using Group Policy Preferences (GPP), highlighting the presence of a vulnerability in the form of GPP encryption (Figure 70). This vulnerability underscores the significance of addressing and mitigating such encryption weaknesses within group policies to prevent unauthorized access to sensitive credentials.

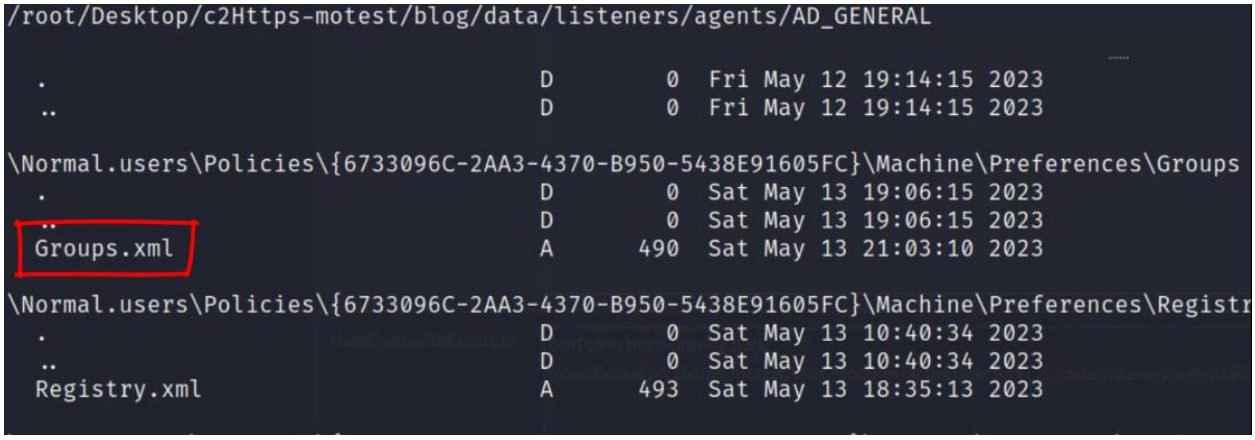

Figure 69 Reaching the Groups.xml

Figure 70 Fetching credentials from Groups.xml

By utilizing the widely known GPP-decrypt.sh script, we successfully cracked the encrypted password (Figure 71). This accomplishment highlights the significance of addressing and mitigating encryption vulnerabilities within Group Policy Preferences (GPP) to prevent unauthorized access to sensitive credentials.

Figure 71 Decrypting the cpassword.

To verify whether WINRM is enabled for the user we obtained through the GPP vulnerability, we employed **crackmapexec**. This powerful tool allows us to perform comprehensive checks and assessments on the user's WINRM configuration, providing insights into potential vulnerabilities or misconfigurations that may exist. By examining the user's WINRM settings, we can further evaluate the overall security posture of the compromised system (Figure 72).

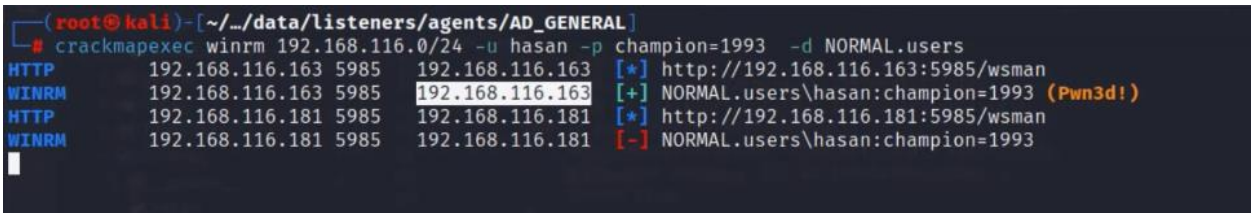

Figure 72 WINRM check

Our analysis, as depicted in Figure 72, revealed that the user "Hasan" possesses WINRM access, indicating membership in the "remote management users" group. Leveraging this information, we initiated a connection using the "evil-winrm" tool (Figure 73). This allows us to exploit the

compromised credentials and gain remote access to the system then execute our C2 framework payload to gain a new agent with Hassan credentials (Figure 74), further highlighting the significance of fortifying access controls and monitoring group memberships within the AD environment.

Figure 73

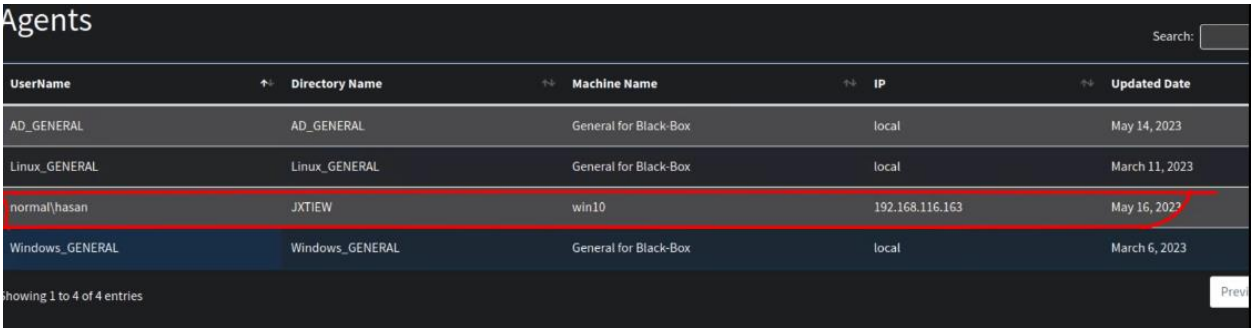

Figure 74

Using our C2 framework, we successfully gained access to Hassan's machine. Through extensive enumeration, we discovered that Hassan is part of a group allowing the RDP protocol. Leveraging this information, we utilized the Lateral_Mov_RDP module, enabling us to establish a remote connection to Hassan's machine via a GUI interface (Figure 75).

Continuing our exploration, we employed Robeaus to dump all TGS tickets of the Service Principal Name (SPN) groups. Our reconnaissance efforts revealed John's TGS ticket, as he belongs to the IT group with potentially elevated privileges (Figure 76). Employing hashcat, we successfully cracked the TGS ticket (Figure 77). Notably, we identified the "force change password" capability enabled within the IT group, particularly when John is associated with it.

Exploiting this finding, we orchestrated a force password change for the administrator, aiming to gain access to the domain admin (Figure 78) (Figure 79). This series of actions underscores the

criticalimportance of implementing robust security measures within Active Directory environments to prevent unauthorized privilege escalation and protect sensitive resources.

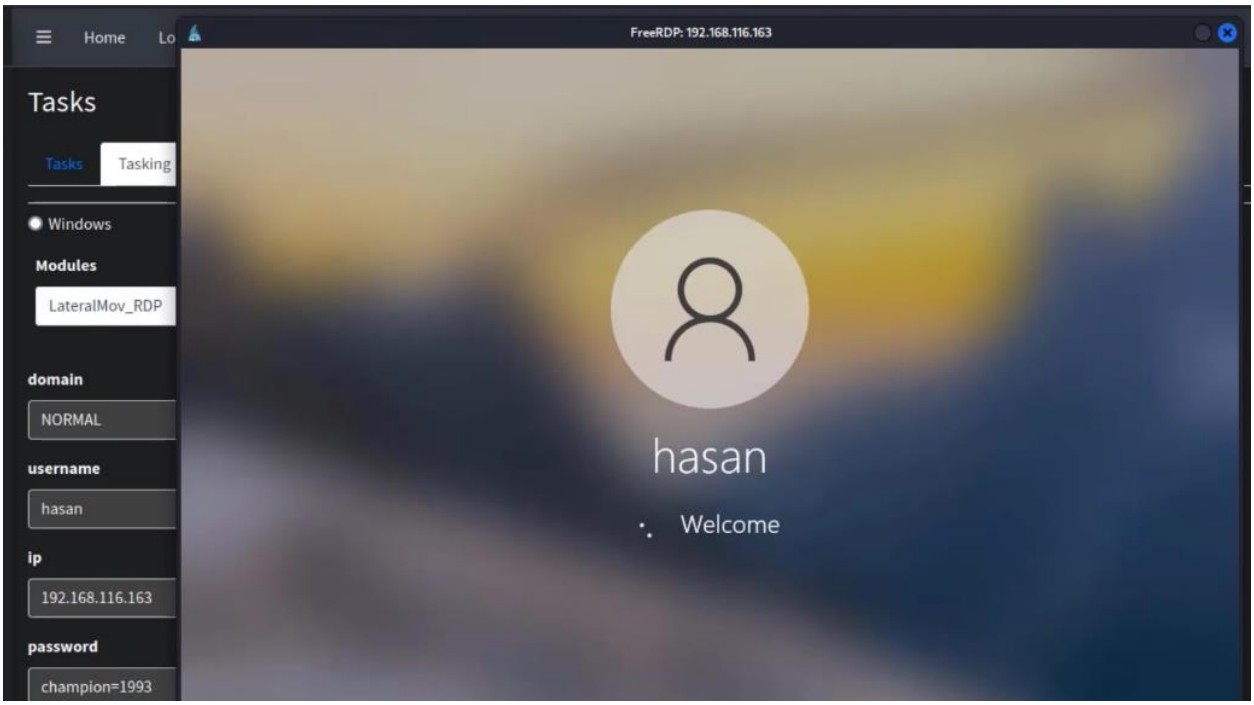

Figure 75

```
Status...........: Cracked
Hash.Mode........: 13100 (Kerberos 5, etype 23, TGS-REP)
Hash.Target......: $krb5tgs$23$*john$NORMAL.USERS$notahacker/LEGIT*$e9 ... 2c28d4
Time.Started.....: Mon May 15 20:24:04 2023 (1 sec)
Time.Estimated ...: Mon May 15 20:24:05 2023 (0 secs)
Kernel.Feature ...: Pure Kernel
Guess.Base.......: File (/usr/share/wordlists/rockyou.txt)
Guess.Queue......: 1/1 (100.00%)
Speed.#1.........:   885.8 kH/s (1.49ms) @ Accel:512 Loops:1 Thr:1 Vec:8
Recovered........: 1/1 (100.00%) Digests (total), 1/1 (100.00%) Digests (new)
Progress.........: 525312/14344385 (3.66%)
Rejected.........: 0/525312 (0.00%)
Restore.Point....: 522240/14344385 (3.64%)
Restore.Sub.#1 ...: Salt:0 Amplifier:0-1 Iteration:0-1
Candidate.Engine.: Device Generator
Candidates.#1....: clavos → carepollo
Hardware.Mon.#1..: Util: 33%

Started: Mon May 15 20:23:59 2023
Stopped: Mon May 15 20:24:06 2023
```

Figure 76

Figure 77

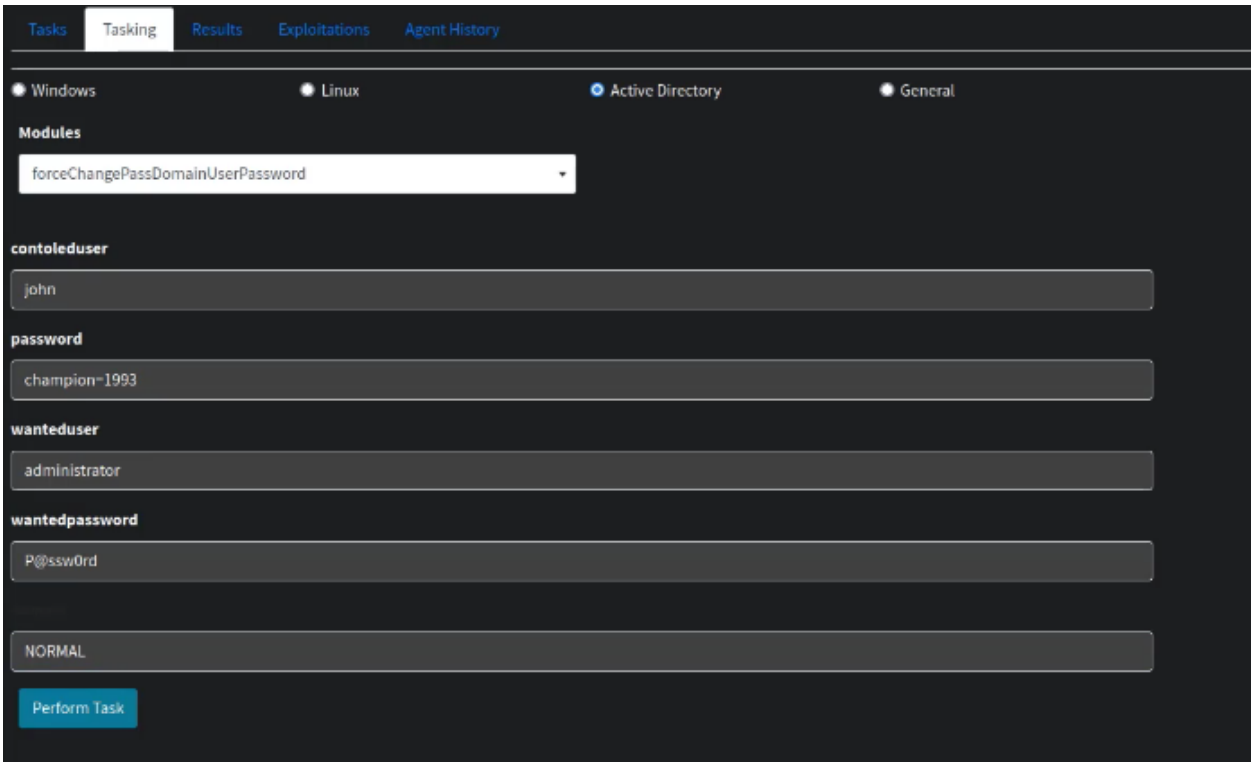

Figure 78

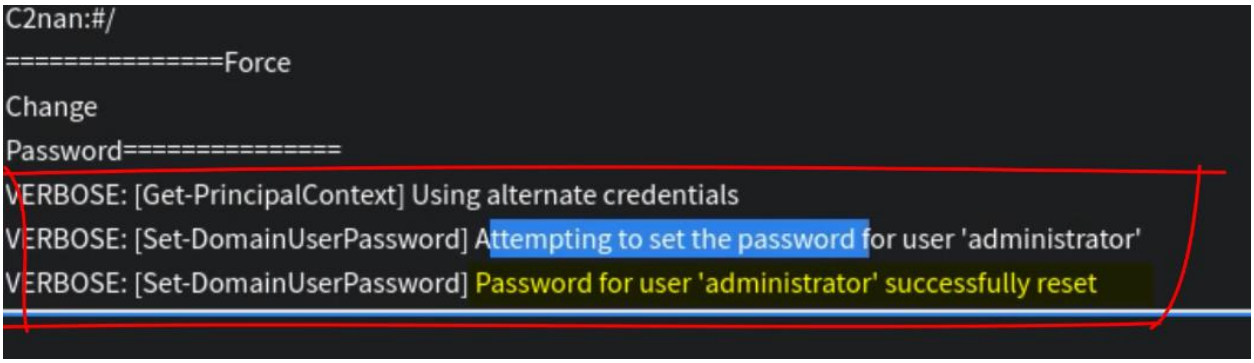

Figure 79

Chapter 10
Conclusion & Future Work

Chapter 10: Conclusion and Future Work

This project analyzed Command and Control (C2) frameworks, demonstrating their practical application in offensive security. The developed prototype showcased its versatility across Windows, Linux, and AD systems, enabling the execution of complete scenarios. The framework's seamless command execution and data collection capabilities contribute to ease the work of security engineer leading to the improvement in the auditing system operation. It was implemented as a web framework using Python-Django, offering a user-friendly interface, automation capabilities, device management, multi-user collaboration, and integration with third-party tools. Although the developed prototype, named C2nan, is limited to a subset of commands and basic scenarios, it serves as a valuable case study.

For future research, there are several areas of improvement and expansion. The focus will be on optimizing and scaling the C2 framework to support various types of agents (e.g., C# and Nim agents), incorporating advanced encryption techniques, payload, and file obfuscation, pivoting, bypass techniques, and addressing additional methodologies and scenarios. These enhancements aim to make the framework more robust and capable of handling a broader range of red-teaming exercises.

References

[1]Armstrong, M. (2022, May 16). *Infographic: The Internet Crime Business is Booming*. Statista Infographics.

[2]Crane, C. (2020, February 21). *42 Cyber Attack Statistics by Year: A Look at the Last Decade - InfoSec Insights*. InfoSec Insights.

[3] *Red Team VS Blue Team: What's the Difference? | CrowdStrike*. (n.d.). crowdstrike.com. https://www.crowdstrike.com/cybersecurity-101/red-team-vs-blue-team/

[4]E. C. (2022, March 28). *Understanding the Five Phases of the Penetration Testing Process*.Cybersecurity Exchange. https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/penetration-testing-phases/

[5] Anderson, C. (2018). "The Red Team: How to Succeed by Thinking Like the Enemy." Publisher.

[6]*Red Team Assessment | Cyber Security | Integrity360*. (n.d.).Red Team Assessment | Cyber Security | Integrity360. https://www.integrity360.com/cyber-security-testing/red-team-assessment

[7]Ross, R. S., Swanson, M., & Stoneburner, G. (2014). "Guide for Conducting Risk Assessments." National Institute of Standards and Technology (NIST) Special Publication 800-30.

[8]James Broad, Andrew Bindner, in Hacking with Kali, 2014.

[9] Jokhio, A. H., Zawoad, S., & Hasan, R. (2020). Comprehensive Study of Command-and-Control Communication Channels in Modern Malware. Computers & Security, 91, 101698.

[10]The framework of Cyber Operations on Command and Control Melanie Bernier1, Sylvain Leblanc2 and Ben Morton2.

[11] Rouse, M. (2018). Command-and-Control Server (C&C Server). TechTarget.

[12] Singh, H., & Sharma, A. (2019). A Comprehensive Study on Command and Control (C2) Mechanisms in Botnet. International Journal of Computer Science and Network Security, 19(1), 77-86.04:36 PM

[13]X. (2020, April 16).*Building a Basic C2*. 0xRick's Blog. https://0xrick.github.io/misc/c2/

[14]C. (2021, April 22). *GitHub - cobbler/Covenant: Covenant is a collaborative .NET C2 framework for red teamers*.GitHub.https://github.com/cobbr/Covenant

[15]*Covenant C2 Framework: The Complete Tutorial*. (2022, December 20). Covenant C2 Framework: The Complete Tutorial. https://blog.netwrix.com/2022/12/16/covenant-c2-tutorial/