Penetration Testing Project

Author: Adir Salinas (code S18)

Class code: 7736

Lecturer: Natali Erez

This document will talk about the Penetration Testing 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 framework, examining each command individually and explaining its specific purpose.

Project Explanation:

This project involves creating a script for comprehensive network device mapping, identifying ports, services, and vulnerabilities. The user defines the network range, after which the program deploys tools like nmap and masscan for scanning and mapping purposes, storing the data in a newly created directory. The script also probes for network vulnerabilities, employing nmap, searchsploit, hydra, and medusa to identify security gaps, such as weak passwords. Finally, the scan summary and findings are presented to the user.

Why Do We Need This Kind of Project?

Risk Mitigation After Breaches:
In the aftermath of security breaches and data leaks, understanding the vulnerabilities in a company's network infrastructure becomes crucial. This project provides a systematic approach to identifying potential security gaps, aiding cyber analysts in understanding the breach and devising strategies to mitigate future risks.

Efficiency in Analysis:
Manual execution of network scans and vulnerability assessments can be labor-intensive and time-consuming, especially in large-scale environments. Automating these processes allows for quicker detection of vulnerabilities and enables cyber analysts to focus their efforts on analyzing results and implementing remediation measures promptly.

Scalability for Complex Networks:
As network infrastructures grow in complexity and scale, manual scanning becomes impractical. Automation provides a scalable solution that can adapt to varying network configurations, allowing for comprehensive assessment of even the largest and most intricate networks.

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 eliminates the need for manual execution of individual commands, significantly reducing the time and effort required to perform network scans and vulnerability assessments. This allows cyber analysts to focus on higher-level tasks, such as analyzing results and developing mitigation strategies.

Adaptability and Flexibility:
Automated scripts can be easily modified and adapted to suit evolving security requirements and changing network environments. This flexibility enables organizations to stay ahead of emerging threats and incorporate new scanning techniques or tools seamlessly into their existing processes.

Project Structure:

The script is structured as a framework designed to accommodate various network analysis needs. Its modular design incorporates multiple menus, each menu responsible for specific tasks, empowering analysts to customize their scans according to their requirements. Analysts can choose from a range of tools, including Nmap, Masscan, searchsploit, hydra, and medusa, based on their specific analysis requirements. This flexibility enables them to leverage the most appropriate tools for each scanning task, maximizing the effectiveness of their analysis efforts.

Key Components of the Framework:

Menu-Based Interface:
The script 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 selecting tools such as Nmap, Masscan, and others, determining the types of ports to include in the scanning process, and configuring scan options for speed and thoroughness.

Reusable Code Components:
The framework incorporates reusable code components, such as functions for executing scans, parsing results, and generating reports. This modular approach promotes code efficiency, maintainability, and reusability, enabling rapid development and easy troubleshooting.

The Company's Response: Managing Security Breaches:

When a company experiences a security breach or suspects a compromise, swift and decisive action is essential to mitigate potential damages and restore security. Here's what typically happens in each phase:

Detection of Breach:

- Detection Mechanisms: Companies often employ various detection mechanisms, including intrusion detection systems (IDS), security information and event management (SIEM) solutions, or even reports from employees or customers.
- Incident Identification: Once a breach is suspected or detected, the organization's incident response team is notified to investigate further.
- Initiating Response: The organization initiates its incident response plan, which may involve isolating affected systems, preserving evidence, and initiating the vulnerability assessment process.

Cybersecurity Response:

- Engagement of Cybersecurity Firm: Depending on the severity and complexity of the breach, the organization may choose to engage the services of a cybersecurity firm with expertise in incident response and digital forensics.
- Forensic Investigation: Cybersecurity experts conduct a thorough investigation to determine the extent of the breach, identify the attacker's tactics, techniques, and procedures (TTPs), and gather evidence for potential legal proceedings.
- Mitigation Measures: Immediate actions are taken to contain the breach and prevent further damage, including patching vulnerabilities, resetting compromised credentials, and isolating affected systems.

Assessment Phase:

- Automated Vulnerability Assessment: As part of the response efforts, the organization leverages automated vulnerability assessment tools to conduct a comprehensive scan of its network infrastructure.
- Identification of Vulnerabilities: The automated tool identifies weaknesses, misconfigurations, and potential entry points exploited by attackers, providing crucial insights into the organization's security posture.

Reporting and Remediation:

- Generation of Detailed Reports: The vulnerability assessment tool generates detailed reports outlining the vulnerabilities detected, their severity levels, and recommendations for remediation.
- Prioritization of Remediation Efforts: Based on the severity and criticality of the vulnerabilities identified, the organization prioritizes remediation efforts to address the most significant risks first.
- Timely Remediation Actions: IT and security teams promptly implement remediation measures, which may include applying patches, updating configurations, and enhancing security controls

Assistance in Cybersecurity Response:

- Expediting Response Efforts: By automating the vulnerability assessment process, the project facilitates a more efficient cybersecurity response, enabling the organization to quickly identify and address vulnerabilities.
- Minimizing Impact: Swift identification and remediation of vulnerabilities help minimize the impact of breaches, reduce potential downtime, and enhance overall resilience against future attacks.
- Continuous Improvement: The insights gained from the vulnerability assessment process inform ongoing security improvements, ensuring that the organization remains vigilant and resilient in the face of evolving cyber threats.

Considering the critical importance of swift response to security breaches, this project can be found as an essential tool for organizations facing cyber threats. By automating the vulnerability assessment process and providing actionable insights, the project significantly enhances the organization's ability to detect, prioritize, and remediate vulnerabilities effectively. Moreover, its structured framework offers flexibility for customization, ensuring that the project can adapt to the specific needs and requirements of users. Ultimately, this project not only delivers valuable data but also empowers organizations with a convenient and efficient approach to bolstering their cybersecurity defenses.

Framework sections and instructions:

To run the script, download the script file to your Linux and type in the terminal "bash PT.Project.sh"

Before the start of the framework, you will need to enter a user password to ensure the script runs smoothly and without any errors. After you insert your password, you will be asked to input a valid network range for the scan. Following that, you'll need to provide a name for the directory where all the data will be saved.

Please ensure that you are not engaging in any illegal activities and that you have the necessary permissions for the scans!

Sections in the Framework and Their Explanations:

- The first section will be responsible for determining the depth of the scan and will ask the user what kind of scan he want to perform Basic/Full:
  Basic – This option will scan the network for TCP, UDP, or both, depending on the analyst's selection. It includes service version detection and identification of weak passwords.
  Full - This option conducts a comprehensive scan of the network, targeting TCP, UDP, or both as per the analyst's choice. It encompasses service version detection, identification of weak passwords, and utilizes the Nmap Scripting Engine (NSE) for vulnerability analysis, including vulnerability identification.

- The second section will be responsible for determining what scanning tool would be used, each with its own advantages depending on the specific requirements.
  It will ask the user what type of scanning tool he want to execute Nmap/Masscan:
  Nmap – is a versatile network scanning tool known for its reliability. Offering thorough and detailed scanning of networks at a more moderate speed, Nmap is renowned for its comprehensive scanning capabilities, scripting engine for automation, and extensive cross-platform support.
  Masscan - known for its high-speed scanning capabilities, though it may sacrifice some reliability and potentially miss certain results due to the speed of the scan. However, it excels in rapid large-scale network reconnaissance, providing customizable scan parameters, efficient resource utilization, and open-source flexibility.

- The third section will be responsible for determining what kind of ports will be scanned and will display a menu for the user to choose according to their requirements TCP/UDP or Both:
  TCP – This option will initiate scanning exclusively for TCP ports.
  UDP - This option will initiate scanning exclusively for UDP ports.
  Both - This option will conduct scans for both TCP and UDP ports concurrently.

- The fourth section will be responsible for determining the speed of the scan to be performed, asking the user to specify their preference Fast or Slow:
  Fast - This option prioritizes speed over accuracy, using Nmap "T4" flag or Masscan "--rate" parameter to maximize scan speed. However, this may sacrifice reliability for faster results.
  Slow - This option prioritizes accuracy and reliability over speed, ensuring thorough and precise results without compromising data integrity.

- The fifth section will be responsible for determining which cracking tool to use in the brute force attack aimed at cracking weak passwords that might be found within the weaknesses identified in the scanning results: Hydra, Medusa, NSE Brute.
  Hydra - A versatile and powerful brute force tool known for its capability to crack passwords across multiple protocols and services.
  Medusa - A command-line tool optimized for fast and parallelized password cracking, supporting multiple protocols and authentication types.
  NSE Brute - NSE Brute is a powerful script available within Nmap's NSE framework, specifically designed for brute force attacks. It enables users to automate the process of attempting various combinations of usernames and passwords to gain unauthorized access to network resources. NSE Brute offers flexibility in targeting different protocols and services, allowing for comprehensive password cracking across diverse network environments.

- The sixth section will be responsible for determining the Password and Username files to be utilized in the brute force attack. To initiate the Brute Force Attack, Usernames and Passwords files are essential.
  The user is presented with a menu offering three options: Auto, Path, Manual.
  Auto – This option automatically generates default usernames and passwords files containing commonly used credentials that will be used within the brute force attack.
  Path – This option allows the user to specify their own username and password files by entering the paths to the files containing the credentials they wish to use in the brute force attack.
  Manual – This option provides the option to manually input a few usernames and passwords on the screen for utilization in the brute force attack.

- The seventh section will be responsible for scanning the target network for vulnerability analysis and conducting searchsploit analysis, then printing reports.
  (THIS PART WILL BE EXECUTED ONLY IF THE USER CHOOSES FULL SCAN!)
  The following output explain what the vulnerability and searchsploit analysis does:
  NSE Vuln - This feature utilizes Nmap's NSE scripts to perform vulnerability analysis on the target network. It identifies potential security vulnerabilities by scanning for known weaknesses in services, software versions, and configurations. NSE Vuln Analysis provides valuable insights into the security posture of the network, helping users prioritize and address vulnerabilities effectively.
  Searchsploit - Searchsploit is a command-line utility used for searching and displaying exploit code from the Exploit Database (exploit-db.com). Searchsploit Analysis leverages this database to identify known exploits related to vulnerabilities discovered during the scan. By providing access to a vast repository of exploit code, Searchsploit Analysis assists users in understanding the potential impact of identified vulnerabilities and aids in the development of mitigation strategies.

- The eighth section will display a menu to the user after completing the entire scanning process, allowing them to inspect all the files created during the scanning process. The options are: OpenIPs, IPRange, WeakCred, LogFile, NSE.
  OpenIPs - Display the list of IP addresses that were discovered to have open ports during the scan and display the scan results.
  IPRange - Display the list of IP addresses range that were scanned during the process.
  WeakCred - Display any weak credentials that were captured and detected during the scan.
  LogFile - Display the created log file containing reports of the most important data.
  NSE - Display the results of the NSE scans that were performed (if the user chooses to perform the NSE scan).
  Within the NSE option, a menu with three options will be displayed if the user selects NSE. This menu will allow the user to inspect the files created by NSE.
    NseBrute - Displays the report generated by NSE Brute if a scan was performed.
    NseVuln - Displays the report generated by NSE Vuln if a scan was performed.

Searchsploit - Displays the report generated by Searchsploit if a scan was performed.

- The final section of the framework will prompt the user to choose whether they want to zip and compress the directory containing all the data or keep it as an uncompressed, normal directory. Yes/No:
  Yes - Select “Yes” if you wish to zip your created directory before exiting…
  No - Select “No” if you wish to keep your created directory as is and unzipped.

Explaining the whole script and each command

The last part of this document will explain everything about command, script and more 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 number of commands that are written in a programming language to perform a specific task or automate a series of tasks. The script executed the commands or tasks by reading them line by line.
In simple words instead of typing each command every time, you are writing all the command you want to execute in a script and when you run the script on a computer then all the commands you 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 be operate and execute only when we call the name of the function if we are not calling 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 how a programmer gives instructions to a computer to perform tasks, make decisions, or control the flow of the program. In simple words, a statement It's 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 in screen the output or data.

Let's talk about few important commands that will be executed in the script:

"echo" - Making space between commands or displaying text on screen.

"clear" - clearing the output terminal screen.

"sleep" - this command used to introduce a delay or pause in the script for a certain period of time. Most of the time there is a number after the command to wait in seconds.

"rm" - The "rm" command, followed by a file name, specifically deletes that file from the file system.

" cd " - Short for "change directory," is used in the script and allows you to navigate between directories and move to a different location.

"grep" - searching for word or sentence inside an output and print them on screen.

"awk" - have a lot of used most of them in the script is to separate certain words or location.

"read" - read command reads the user's input and assigns it to the variable, using the flag (-p) after the command for printing text on screen before the input.

"mkdir" - The "mkdir" command, used in the command line interface, creates directories (folders) within the file system. By providing directory names as arguments, "mkdir" generates these directories if they do not already exist. This command is indispensable for organizing files and managing directory structures efficiently.

"nmap" - is an open-source network scanning tool used for discovering devices, services, and vulnerabilities on computer networks.

"Masscan" - known for its high-speed scanning capabilities, though it may sacrifice some reliability and potentially miss certain results due to the speed of the scan.

"sudo" - command is used to execute commands with elevated privileges. and it allows a permitted user to execute a command as the superuser.

"sudo apt-get install" - command to install different packages or applications.

"> /dev/null 2>&1" - this command used in addition to another command to execute the command quietly without displaying any output or errors.

Functions names, their order and short description:

Function that will ask for a Network Range to Scan where all the scans will be performed on: IP_Range.

Function that will ask for a name for the directory where all the data will be saved: OutDir.

Function that will verify whether essential tools are installed and, if not, installs them: ToolCheck.

Function that will be responsible for determining the depth of the scan and will ask the user what kind of scan he want to perform Basic/Full: CaseChoice.

Function will scan the network for TCP, UDP, or both, depending on the analyst's selection. It includes service version detection and identification of weak passwords: Basic.

Function that will be responsible for utilizing the Nmap Scripting Engine (NSE) for vulnerability analysis, including Searchsploit and vulnerability identification: Full.

Function will be responsible for conducting nmap scan: NmapScan.

Function that will be responsible for conducting masscan scan: MasscanScan.

Function that will be responsible for conducting scans for TCP ports: TcpScan.

Function that will be responsible for conducting scans for UDP ports: UdpScan.

Function that will be responsible for determining the speed of the TCP ports scan: TcpScan_Slow, TcpScan_fast.

Function that will be responsible for determining the speed of the UDP ports scan: UdpScan_Slow, UdpScan_fast.

Function that will be responsible for determining the speed of both ports scan: BothScan_Slow, BothScan_fast.

Function that will be responsible for conducting the brute force on the found scan results: BruteForce.

Function that will be responsible for conducting Brute force attack using different cracking tools: Hydra, Medusa, Nse.

Function that will be responsible for conducting Brute force attack using an automatic created files of passwords and usernames of the most known ones, then printing reports: AutoFiles.

Function that will be responsible for conducting Brute force attack using a file of passwords and usernames from the user, then printing reports: UserFiles.

Function that will be responsible for conducting Brute force attack using passwords and usernames from the user, then printing reports: UserWordsLog.

Function that will be responsible for determining the Password and Username files to be utilized in the brute force attack. To initiate the Brute Force Attack, Usernames and Passwords files are essential. The user is presented with a menu offering three options: Auto, Path, Manual: LoginChoice.

Function that will be responsible for determining which cracking tool to use in the brute force attack aimed at cracking weak passwords that might be found within the weaknesses identified in the scanning results: Hydra, Medusa, NSE Brute: BFChoice.

Function that will be responsible for determining what scanning tool would be used, each with its own advantages depending on the specific requirements. It will ask the user what type of scanning tool he wants to execute Nmap/Masscan: NmapMasscan.

Function that will be responsible for scanning the target network for vulnerability analysis, then printing reports: NseVuln.

Function that will be responsible for conducting searchsploit analysis, then printing reports: Searchsploit.

Function that will display a menu to the user after completing the entire scanning process, allowing them to inspect all the files created during the scanning process. The options are: OpenIPs, IPRange, WeakCred, LogFile, NSE: ResualtMenu.

Function that will be responsible for creating the Log file containing most of the information that was created: LogFile.

Braking down the script to parts and explain each part in details

(Basic command that was explained before will not have description about them)

Inside the functions there will be other functions names that getting called when needed!

Commands that will be used often:

Cd $***/$***/$***

Whenever the cd command is being used in the script with various variables or names, it is changing the location of the current directory to the one specified by those variables or different names.

echo > (file name)

When using the command “echo” with a single “>” and after it a file name, this command empties the content of the specified file (or creates an empty file if it doesn't exist).
It uses a single > symbol, known as output redirection, to overwrite the contents of the file.

echo >> (file name)

When using the command “echo” with a double “>” and after it a file name, this command appends the output of the echo command to the end of the specified file. it uses double “>>” symbols for output redirection, ensuring that the existing content in the file is preserved, and adding a new content without overwriting or deleting the output inside.

Break

The "break" statement is used to exit a loop or a switch statement prematurely.
It causes the program to immediately exit the nearest enclosing loop or switch block and continue executing the code after the loop or switch block.

read -p “<some text>”

The "read" command in combination with the "-p" flag allows the user to prompt for input with a specified message or text. This facilitates interactive shell scripts by displaying the specified text to the user before awaiting input.
The user's input is then stored in a variable for further processing within the script.

echo “${bold}/ ${red}”

In shell scripting, "${bold}" is used to apply bold formatting to text when echoed to the terminal. It's part of a broader practice of using escape sequences or variables to control text formatting, including colors for example “${red}”. This allows scripts to generate visually appealing and informative output, enhancing readability and user experience.

At the beginning of the script, we set up some variables to make the text look fancier. These variables help us style the text by making it bold or changing its color. Let's take a closer look at each of these commands and what they do.

```
bold=$(tput bold) # sets the variable "bold" to the escape sequen
normal=$(tput sgr0) # sets the variable "normal" to the escape se
green=$(tput setaf 2) # sets the variable "green" to the escape s
blue=$(tput setaf 4) # sets the variable "blue" to the escape seq
red=$(tput setaf 1) # sets the variable "red" to the escape seque
purple=$(tput setaf 5) # sets the variable "purple" to the escape
yellow=$(tput setaf 3) # sets the variable "yellow" to the escape
```

bold=$(tput bold)

The command sets the variable 'bold' to the escape sequence for bold text formatting using the 'tput' terminal utility, enabling the styling of text as bold in the script

normal=$(tput sgr0)

The command sets the variable ' normal ' that uses the 'tput' terminal utility to assign the escape sequence for resetting all text attributes, The “sgr0” in the command represents the "Select Graphic Rendition" escape sequence that resets all text attributes to their default values, allowing text to return to its normal appearance in the script.

green=$(tput setaf 2)

The command sets the variable 'green' that uses the 'tput' command to assign the escape sequence for green text color. The 'setaf' command is a part of the 'tput' utility, representing 'Set ANSI Foreground,' and the '2' parameter specifies the green color, enabling subsequent text in the script to appear in green.

blue=$(tput setaf 4)

The command employs the 'tput' utility to set the variable 'blue' with the escape sequence for changing text color to blue. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '4' specifies the blue color, enabling subsequent text in the script to appear in blue.

red=$(tput setaf 1)

The command utilizes the 'tput' utility to set the variable 'red' with the escape sequence for changing text color to red. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '1' specifies the red color, enabling subsequent text in the script to appear in red.

purple=$(tput setaf 5)

The command 'purple=$(tput setaf 5)' employs the 'tput' utility to set the variable 'purple' with the escape sequence for changing text color to purple. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '5' specifies the purple color, enabling subsequent text in the script to appear in purple.

yellow=$(tput setaf 3)

The command 'yellow=$(tput setaf 3)' employs the 'tput' utility to set the variable 'yellow' with the escape sequence for changing text color to yellow. In the 'setaf' command, representing 'Set ANSI Foreground,' the parameter '3' specifies the yellow color, enabling subsequent text in the script to appear in yellow.

The following command ensures user authentication

```
clear
echo "[!] Before We Start, Please Insert Your Pa
echo
sleep 2

sudo -kv # updates the user's cached credentials

location=$(pwd) # captures the current working d
```

sudo -kv

The sudo -kv command updates the user's cached credentials, requesting the user's password, if necessary, without executing any specific command. Used within the script to ensure that previously entered sudo passwords won't be prompted again for subsequent sudo commands.

location=$(pwd)

The command sets the variable "location" to the present working directory using the (pwd) command, representing the location path of the current directory in the script.

Within the rest of the script the "location" variable will be used often.
Along with the "cd" command, it means changing the location to the directory stored in that variable.
(where the script file is located).

Function that will ask for a Network Range to Scan where all the scans will be performed on:

```
function IP_Range () # This function will ask for a Network Range to Scan where all the scans will be preform on.
{

    cd $location
    clear
    echo
    while true
    do
        read -p "[?] Please Enter a Network Range to Scan: " Range
        ipchk=$(echo $Range | grep -oE '([0-9]{1,3}\.){3}[0-9]{1,3}')
        if [ -z $(echo $ipchk |egrep "^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$") ]
        then
            sleep 2
            echo
            echo "[!] The Provided IP Address Range Is Invalid! Please try again... "
            echo
            sleep 2
        else
            rm error > /dev/null 2>&1
            test=$(nmap $Range -sL 2> error 1> IPRangeFile)
            if [ ! -z "$(cat error)" ]
            then
                echo
                echo "[!] The Provided IP Address Range Is Invalid! Please try again... "
                echo
            else
                rm error > /dev/null 2>&1
                break
            fi

        fi
    done
```

Within the function, there is a while loop that will continue looping until the user enters a valid IP address range.

ipchk=$(echo $Range | grep -oE '([0-9]{1,3}\.){3}[0-9]{1,3}')

In this command, the variable "ipchk" is assigned the result of a grep command applied to the value stored in the variable "$Range". The grep command searches for IP addresses within the string stored in "$Range", using a regular expression pattern to match IPv4 addresses. The matched IP addresses are then stored in the "ipchk" variable for further processing within the script.

if [-z $(echo $ipchk |egrep "^((25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$")]

This if conditional statement checks if the variable "ipchk" contains a valid IPv4 address using a regular expression pattern. The "-z" flag checks if the length of the string resulting from the grep command is zero, indicating that no valid IP address was found.
If no valid IP address is found, the condition evaluates to true, and the script proceeds to the "then" block, if a valid IPv4 address is found, the condition evaluates to false, and the script proceeds to the "else" block.

rm error

The "rm" command, followed by the file name "error", specifically deletes the file named "error" from the file system.

test=$(nmap $Range -sL 2> error 1> IPRangeFile)

In this command, the variable "test" is assigned the result of running the "nmap" command with the specified range and options. The "-sL" flag instructs nmap to only list the hosts without scanning, while "2> error" redirects standard error (stderr) to a file named "error", capturing any error messages. Similarly, "1> IPRangeFile" redirects standard output (stdout) to a file named "IPRangeFile", storing the list of hosts generated by nmap.

if [! -z "$(cat error)"]

This conditional statement checks if the file "error" is not empty by using the "cat" command within the command substitution "$(cat error)".
If the file contains any content (indicating an error message captured by nmap), the condition evaluates to true, prompting the script to execute the commands in the "then" block. Conversely, if the file is empty, the condition evaluates to false, and the script proceeds to execute the commands in the "else" block.

Function that will ask for a name for the directory where all the data will be saved

```
function OutDir () # This function will ask for a name for the directory where all the data will be saved.
{
    echo
    echo
    while true;
    do
        read -p "[?] Please Enter a Name for The Scan Output Directory: " DirName
        if [ -d "$DirName" ];
        then
            echo
            echo "[!] Directory '$DirName' already exists! Please choose a different name."
            echo
            sleep 2
        else
            break
        fi
    done
    cd $location
    mkdir $DirName > /dev/null 2>&1
    mv IPRangeFile $location/$DirName
    sleep 2
}
OutDir
```

Within the function, there is a while loop that will continue looping until the user enters a valid directory name that doesn't exist already.

if [-d "$DirName"]

This conditional statement checks if the variable "$DirName" corresponds to an existing directory using the "-d" flag. If "$DirName" refers to an existing directory, the condition evaluates to true and moving to then statement and if not move to else statement.

Function that will verify whether essential tools are installed and, if not, installs them

```
function ToolCheck () # This function will check if the essen
{
    for i in hydra medusa exploitdb
    do
        if ! command -v $i &> /dev/null
        then
            echo
            echo
            echo "${bold}[?] Please wait while we install ess
            sudo apt-get -qq -y install $i > /dev/null 2>&1
        fi
    done
}
ToolCheck
```

for i in hydra medusa exploitdb

This is a "for" loop that iterates over the element's "hydra", "medusa", and "exploitdb" sequentially, assigning each element to the variable "i" in turn for further processing within the loop.

if ! command -v $i

This conditional statement checks if the command represented by the variable "$i" is not found in the system's PATH. If the command is not found, the condition evaluates to true, indicating that the command is not available for execution.

sudo apt-get -qq -y install $i

In this command, "sudo apt-get" is used to install packages from the system's package repository with elevated privileges. The options "-qq" suppress the progress output and "-y" automatically confirms prompts, allowing for non-interactive installation.
The variable "$i" represents the package name to be installed, which is passed as an argument to the command.

Function will scan the network for TCP, UDP, or both, depending on the analyst's selection. It includes service version detection and identification of weak passwords

```
function Basic () # This function will scan the network for TCP, UDP, or both, depending on the analyst's selection. It i
{
    function NmapScan () # This Fucntion will be responsible for conducting nmap scan.
    {
        clear
        cd $location/$DirName
        mkdir NmapScanRes > /dev/null 2>&1

        function TcpScan () # This Fucntion will be responsible for conducting nmap scan for tcp ports
        {
            function TcpScan_fast () # This Fucntion will be responsible for conducting nmap scan for tcp ports and in fa
            {
                for IP in $(cat IPRangeFile |awk '{print $(NF)}'|grep ^[0-9])
                do
                    echo
                    echo "                                        |"
                    echo "                                        ▼  "
                    echo "                                  Scanning $IP "
                    sudo nmap -p- -sV -T4 -Pn $IP > NmapScanRes/$IP
                done
                sleep 2.5
            }
            function TcpScan_Slow () # This Fucntion will be responsible for conducting nmap scan for tcp ports and in nc
            {
                for IP in $(cat IPRangeFile |awk '{print $(NF)}'|grep ^[0-9])
                do
                    echo
                    echo "                                        |"
                    echo "                                        ▼  "
                    echo "                                  Scanning $IP "
                    sudo nmap -p- -sV -Pn $IP > NmapScanRes/$IP
                done
                sleep 2.5
            }
```

Within the Basic function, there are additional functions, each responsible for different actions discussed earlier in this PDF, along with their respective responsibilities.

for IP in $(cat IPRangeFile |awk '{print $(NF)}'| grep ^[0-9])

This "for" loop iterates over each IP address listed in the "IPRangeFile" file, extracting the last field of each line using "awk '{print $(NF)}'", which represents the IP address.
The "grep ^[0-9]" command filters out any lines that do not start with a digit, ensuring that only valid IP addresses are processed within the loop.

sudo nmap -p- -sV -T4 -Pn $IP > NmapScanRes/$IP

In this command, "sudo nmap" is used to perform a port scan on the specified IP address with elevated privileges. The options "-p-" scan all ports, "-sV" enable version detection, "-T4" sets the timing template to aggressive, and "-Pn" skips host discovery. The output of the scan is redirected to a file named after the IP address in the directory "NmapScanRes/ ".

sudo nmap -p- -sV -Pn $IP > NmapScanRes/$IP

In this command, "sudo nmap" is used to perform a port scan on the specified IP address with elevated privileges. The options "-p-" scan all ports, "-sV" enable version detection and "-Pn" skips host discovery. The output of the scan is redirected to a file named after the IP address in the directory "NmapScanRes/ ".

While statement that will be responsible for determining the speed of the scan to be performed, asking the user to specify their preference Fast or Slow:

```
while true
do
    clear
    echo
    echo "${bold}[»] Please Choose the Speed of Scan You Want to Preform:"
    echo
    echo "${yellow} F) Fast - Prioritizes speed over accuracy, providing results quickly but with a
    echo " S) Slow - Emphasizes accuracy and reliability over speed, ensuring thorough and precise
    echo
    echo "${red}${bold}    +-----------------------------------------------------------+"
    echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
    echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
    echo "    +-----------------------------------------------------------+  ${normal}"
    echo
    echo "${bold}     ┌─[Enter Your Choice]"
    read -p  "     └──────► " choice
    Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')
    case $choice in

        f|fast)
            echo "             └──────►${purple} You Have Chosen to Scan in Fast Mode!    ${normal}"
            sleep 3
            TcpScan_fast
            break
        ;;
        s|slow)
            echo "             └──────►${purple} You Have Chosen to Scan in Slow Mode!    ${normal}"
            sleep 3
            TcpScan_Slow
            break
        ;;
        99)
            echo "             └──────► ${red}Going Back to Previous Menu${normal}${bold} ─┐"
            echo "                                                  ◄──┘     ${normal}"
            sleep 3
            NmapMasscan
            break
        ;;
        ..|exit)
            echo "             └──────►${red} Exiting the framework... ${normal}"
            sleep 2
            exit
        ;;
        *)
            echo "             └──────►${green} Invalid choice! Please try again... ${normal}"
            sleep 3
        ;;

    esac
done
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

Function that will be responsible for conducting nmap scan for UDP ports

```
function UdpScan () # This Fucntion will be responsible for conducting masscan scan for udp ports
{
    function UdpScan_fast () # This Fucntion will be responsible for conducting nmap scan for udp
    {
        for IP in $(cat IPRangeFile |awk '{print $(NF)}'|grep ^[0-9])
        do
            echo
            echo "                                        |"
            echo "                                        ▼  "
            echo "                                  Scanning $IP "
            sudo nmap -sU -sV -T4 -Pn $IP > NmapScanRes/$IP
        done
        sleep 2.5
    }
    function UdpScan_Slow () # This Fucntion will be responsible for conducting nmap scan for udp
    {
        for IP in $(cat IPRangeFile |awk '{print $(NF)}'|grep ^[0-9])
        do
            echo
            echo "                                        |"
            echo "                                        ▼  "
            echo "                                  Scanning $IP "
            sudo nmap -sU -sV -Pn $IP > NmapScanRes/$IP
        done
        sleep 2.5
    }
```

for IP in $(cat IPRangeFile |awk '{print $(NF)}'| grep ^[0-9])

This "for" loop iterates over each IP address listed in the "IPRangeFile" file, extracting the last field of each line using "awk '{print $(NF)}'", which represents the IP address.
The "grep ^[0-9]" command filters out any lines that do not start with a digit, ensuring that only valid IP addresses are processed within the loop.

sudo nmap -sU -sV -T4 -Pn $IP > NmapScanRes/$IP

In this command, "sudo nmap" is used to perform a port scan on the specified IP address with elevated privileges. The options "-sU" scan UDP ports, "-sV" enable version detection, "-T4" sets the timing template to aggressive, and "-Pn" skips host discovery.
The output of the scan is redirected to a file named after the IP address in the directory "NmapScanRes/ ".

sudo nmap -sU -sV -Pn $IP > NmapScanRes/$IP

In this command, "sudo nmap" is used to perform a port scan on the specified IP address with elevated privileges. The options "-sU" scan UDP ports, "-sV" enable version detection and "-Pn" skips host discovery.
The output of the scan is redirected to a file named after the IP address in the directory "NmapScanRes/ ".

While statement that will be responsible for determining the speed of the scan to be performed, asking the user to specify their preference Fast or Slow:

```
while true
do
    clear
    echo
    echo "${bold}[»] Please Choose the Speed of Scan You Want to Preform:"
    echo
    echo "${yellow} F) Fast - Prioritizes speed over accuracy, providing results quickly but with a
    echo " S) Slow - Emphasizes accuracy and reliability over speed, ensuring thorough and precise
    echo
    echo "${red}${bold}    +-----------------------------------------------------------+"
    echo "     |  99) To Return to the Previous Menu, Simply type '99'  |"
    echo "     |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
    echo "     +--------------------------------------------------------+  ${normal}"
    echo

    echo "${bold}     ┌─[Enter Your Choice]"
    read -p  "     └─────► " choice
    Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')
    case $choice in

        f|fast)
            echo "             └─────►${purple} You Have Chosen to Scan in Fast Mode!    ${normal}"
            sleep 3
            UdpScan_fast
            break
        ;;
        s|slow)
            echo "             └─────►${purple} You Have Chosen to Scan in Slow Mode!    ${normal}"
            sleep 3
            UdpScan_Slow
            break
        ;;
        99)
            echo "             └─────► ${red}Going Back to Previous Menu${normal}${bold} ─┐"
            echo "                                                  ◄─┘    ${normal}"
            sleep 3
            NmapMasscan
            break
        ;;
        ..|exit)
            echo "             └─────►${red} Exiting the framework... ${normal}"
            sleep 2
            exit
        ;;
        *)
            echo "             └─────►${green} Invalid choice! Please try again... ${normal}"
            sleep 3
        ;;

    esac
done
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

This Function will be responsible for conducting nmap scan for both ports.

```
function BothScan () # This Fucntion will be responsible for conducting nmap scan for both ports
{
    function BothScan_fast () # This Fucntion will be responsible for conducting nmap scan for b
    {
        for IP in $(cat IPRangeFile |awk '{print $(NF)}'|grep ^[0-9])
        do
            echo
            echo "                                        |"
            echo "                                        V  "
            echo "                                  Scanning $IP "
            sudo nmap -p- -sU -sT -sV -T4 -Pn $IP > NmapScanRes/$IP
        done
        sleep 2.5
    }
    function BothScan_Slow () # This Fucntion will be responsible for conducting nmap scan for b
    {
        for IP in $(cat IPRangeFile |awk '{print $(NF)}'|grep ^[0-9])
        do
            echo
            echo "                                        |"
            echo "                                        V  "
            echo "                                  Scanning $IP "
            sudo nmap -p- -sU -sT -sV -Pn $IP > NmapScanRes/$IP
        done
        sleep 2.5
    }
```

for IP in $(cat IPRangeFile |awk '{print $(NF)}'| grep ^[0-9])

This "for" loop iterates over each IP address listed in the "IPRangeFile" file, extracting the last field of each line using "awk '{print $(NF)}'", which represents the IP address.
The "grep ^[0-9]" command filters out any lines that do not start with a digit, ensuring that only valid IP addresses are processed within the loop.

sudo nmap -p- -sU -sT -sV -T4 -Pn $IP > NmapScanRes/$IP

In this command, "sudo nmap" conducts a comprehensive scan on the specified IP address with elevated privileges.
The options "-p-" scan all ports, "-sU" enable UDP port scanning, "-sT" enable TCP port scanning, "-sV" enable version detection, "-T4" sets the timing template to aggressive, and "-Pn" skips host discovery.
The results of the scan are then redirected to a file named after the IP address in the directory "NmapScanRes/".

sudo nmap -p- -sU -sT -sV -Pn $IP > NmapScanRes/$IP

In this command, "sudo nmap" performs a thorough scan on the specified IP address with elevated privileges.
The options "-p-" scan all ports, "-sU" enables UDP port scanning, "-sT" enables TCP port scanning, "-sV" enables version detection, and "-Pn" skips host discovery.
The results of the scan are then saved to a file named after the IP address in the directory "NmapScanRes/".

While statement that will be responsible for determining the speed of the scan to be performed, asking the user to specify their preference Fast or Slow:

```
while true
do
    clear
    echo
    echo "${bold}[»] Please Choose the Speed of Scan You Want to Preform:"
    echo
    echo "${yellow} F) Fast - Prioritizes speed over accuracy, providing results quickly but with a
    echo " S) Slow - Emphasizes accuracy and reliability over speed, ensuring thorough and precise
    echo
    echo "${red}${bold}    +-----------------------------------------------------------+"
    echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
    echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
    echo "    +-----------------------------------------------------------+  ${normal}"
    echo
    echo "${bold}    ┌─[Enter Your Choice]"
    read -p  "    └──────► " choice

    Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')
    case $choice in

        f|fast)
            echo "            └──────►${purple} You Have Chosen to Scan in Fast Mode!    ${normal}"
            sleep 3
            BothScan_fast
            break
        ;;
        s|slow)
            echo "            └──────►${purple} You Have Chosen to Scan in Slow Mode!    ${normal}"
            sleep 3
            BothScan_Slow
            break
        ;;
        99)
            echo "            └──────► ${red}Going Back to Previous Menu${normal}${bold} ─┐"
            echo "                                                  ◄─┘    ${normal}"
            sleep 3
            NmapMasscan
            break
        ;;
        ..|exit)
            echo "            └──────►${red} Exiting the framework... ${normal}"
            sleep 2
            exit
        ;;
        *)
            echo "            └──────►${green} Invalid choice! Please try again... ${normal}"
            sleep 3
        ;;

    esac
done
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

While statement that will be responsible for determining what kind of ports will be scanned and will display a menu for the user to choose according to their requirements TCP/UDP or Both:

```
while true
do
    clear
    echo
    echo "${bold}[»] Which type of port scan do you want to perform: TCP, UDP, or both?"
    echo
    echo "${yellow} T) TCP - Will initiate a scanning for TCP ports only. "
    echo " U) UDP - Will initiate a scanning for UDP ports only."
    echo " B) Both - Will initiate a scanning for both TCP and UDP ports. ${normal}"
    echo
    echo "${red}${bold}    +-----------------------------------------------------------+"
    echo "     |  99) To Return to the Previous Menu, Simply type '99'  |"
    echo "     |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
    echo "     +-----------------------------------------------------------+ ${normal}"
    echo

    echo "${bold}     ┌─[Enter Your Choice]"
    read -p  "     └─────► " UserChoice
    Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

    case $Choice in

        t|tcp)
            echo "             └─────►${purple} You Have Chosen to Scan Only TCP Ports!   ${normal}"
            sleep 3
            TcpScan
            break

        ;;
        u|udp)
            echo "             └─────►${purple} You Have Chosen to Scan Only UDP Ports!  ${normal}"
            sleep 3
            UdpScan
            break
        ;;
        b|both)
            echo "             └─────►${purple} You Have Chosen to Scan Both Kind of Ports!  ${normal}"
            sleep 3
            BothScan
            break
        ;;
        99)
            echo "             └─────►${red} Going Back to Previous Menu ${normal}${bold}─┐"
            echo "                                                       └─┘ ${normal}"
            sleep 3
            NmapMasscan
            break
        ;;
        ..|exit)
            echo "             └─────►${red} Exiting the framework... ${normal}"
            sleep 2
            exit
        ;;
        *)
            echo "             └─────►${green} Invalid choice! Please try again...  ${normal}"
            sleep 3
        ;;

    esac
done
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

This Function will be responsible for conducting the brute force on the found scan results, then printing reports.

The function within the BruteForce function will be responsible for conducting Brute force attack using hydra using an automatic created files of passwords and usernames of the most known ones, then printing reports.

```
function BruteForce () # This Fucntion will be responsible for conducting the brute force on the found scan resualts, then printing reports.
{
    function Hydra () # This Fucntion will be responsible for conducting Brute force attack using hydra, then printing reports.
    {
        cd $location/$DirName
        mkdir WeakPassRes
        cd $location/$DirName/NmapScanRes

        function AutoFiles () # This Fucntion will be responsible for conducting Brute force attack using hydra using an automatic created files of passwords and usernames of the most known ones, then printing reports.
        {
            cd $location/$DirName
            mkdir Login_File > /dev/null 2>&1
            cd Login_File
            echo -e "root\nadmin\nuser\ntest\n1234\nguest\n123\n12345\n123456\n1234567\n12345678\npassword\nPassword\n1234567aA\nPassw0rd\nftp\nadministrator\nssh\npassword1\n123456789" > password.lst
            echo -e "root\nadmin\nuser\nguest\nftp\nadministrator\nssh" >> namelist.txt
            cd $location/$DirName/NmapScanRes
            echo
            echo "[*] Please wait while we initiate the Brute Force attack..."
            sleep 1
            echo "[!] Any results that have been captured will be displayed on the screen!"
            echo
            cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line;
            do

                ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)
                service=$(echo "$line" | awk '{print $3}')
                port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

                hydra -L $location/$DirName/Login_File/namelist.txt -P $location/$DirName/Login_File/password.lst  $service://$ip:$port 2>/dev/null |grep -wi "login:" -B1 |tee -a $location/$DirName/WeakPassRes/"Hydra $
                echo

            done
        }
```

echo -e "root\nadmin\nuser\ntest\n1234\...\....\....\...\npassword1\n123456789" > password.lst

This command creates a file named "password.lst" and populates it with a list of common passwords separated by newline characters.
The "-e" flag enables interpretation of backslash escapes, allowing for the inclusion of newline characters ("\n") to separate each password entry.

echo -e "root\nadmin\nuser\nguest\nftp\nadministrator\nssh" >> namelist.txt

This command appends a list of usernames to the file "namelist.txt".
Each username is separated by a newline character ("\n"), specified by the "-e" flag with the "echo" command.
The ">>" operator appends the output to the end of the existing file "namelist.txt".

cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line

This command pipes the output of "cat *" (which concatenates the contents of all files in the current directory) to two "grep" commands.
The first "grep" filters for lines containing the case-insensitive word "open".
The second "grep" command further filters the output, searching for lines containing any of the specified protocols ("ftp", "ssh", "rdp", "telnet").
The results are then processed line by line in a "while" loop, where each line is stored in the variable "line".

ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)

This command extracts the IP address from each line stored in the variable "line". "echo "$line"" prints the current line, "awk '{print $1}'" extracts the first field (presumably the IP address) from the line. "cut -d ':' -f 1" separates the IP address from any port number by using ":" as a delimiter and selects the first field before ":".
The resulting IP address is stored in the variable "ip".

service=$(echo "$line" | awk '{print $3}')

This command extracts the service name from each line stored in the variable "line" by printing the third field using the "awk" command.

port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

This command extracts the port number from each line stored in the variable "line" by first using ":" as the field separator to isolate the port section, then using "/" as the field separator to extract the port number.

hydra -L $location/../namelist.txt -P $location/../password.lst $service://$ip:$port |grep -wi "login:" -B1 |tee -a $location/…/"Hydra $ip"

This command utilizes Hydra to perform a brute force attack, attempting login combinations from specified username and password lists. The "-L" flag specifies the username list file, "-P" specifies the password list file, and "$service://$ip:$port" specifies the target service, IP address, and port. The "grep" command filters for lines containing "login:" along with the preceding line. Finally, "tee -a" appends the output to a file named after the IP address in the "WeakPassRes" directory.

This Function will be responsible for conducting Brute force attack using hydra using a file of passwords and usernames from the user, then printing reports.

```
function UserFiles () # This Fucntion will be responsible for conducting Brute force attack using hydra using a files of passwords and usernames from the us
{
    clear
    echo
    while true
    do
        read -p "[»] Please Enter a Path to a File That Contains Usernames That You Wish to Use: " file_name_path
        if [ -f "$file_name_path" ]
        then
            sleep 1
            echo
            echo
            echo "[✓] File is Exists and Valid to Use. Progressing... "
            echo
            echo
            break
        else
            echo
            echo
            sleep 2
            echo "[!] File Not Found, Please Try Again... "
            echo
            echo
            sleep 2
        fi
    done

    while true
    do
        read -p "[»] Please Enter a Path to a File That Contains Passwords That You Wish to Use: " file_pass_path
        if [ -f "$file_pass_path" ]
        then
            sleep 1
            echo
            echo
            echo "[✓] File is Exists and Valid to Use. Progressing... "
            echo
            echo

            echo
            break
        else
            echo
            echo
            sleep 2
            echo "[!] File Not Found, Please Try Again... "
            echo
            echo
            sleep 2
        fi
    done
    echo
    echo
    echo "[√] Thank you! Your data input has been successfully recorded! "
    sleep 1
    echo
    echo "[*] Please wait while we initiate the Brute Force attack..."
    sleep 1
    echo "[!] Any results that have been captured will be displayed on the screen!"
    echo
    cd NmapScanRes
    cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line;
    do

        ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)
        service=$(echo "$line" | awk '{print $3}')
        port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

        hydra -L $file_name_path -P $file_pass_path $service://$ip:$port 2>/dev/null |grep -wi "login:" -B1 |tee -a $location/$DirName/WeakPassRes/"Hydra $ip"
        echo

    done
}
```

if [-f "$file_name_path"]

This conditional statement checks if the file specified by the variable "$file_name_path" exists using the "-f" flag. If the file exists, the condition evaluates to true.

if [-f "$file_pass_path"]

This conditional statement checks if the file specified by the variable "$file_pass_path" exists using the "-f" flag. If the file exists, the condition evaluates to true.

cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line

This command pipes the output of "cat *" (which concatenates the contents of all files in the current directory) to two "grep" commands.
The first "grep" filters for lines containing the case-insensitive word "open".
The second "grep" command further filters the output, searching for lines containing any of the specified protocols ("ftp", "ssh", "rdp", "telnet").
The results are then processed line by line in a "while" loop, where each line is stored in the variable "line".

ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)

This command extracts the IP address from each line stored in the variable "line".
"echo "$line"" prints the current line, "awk '{print $1}'" extracts the first field (presumably the IP address) from the line. "cut -d ':' -f 1" separates the IP address from any port number by using ":" as a delimiter and selects the first field before ":".
The resulting IP address is stored in the variable "ip".

service=$(echo "$line" | awk '{print $3}')

This command extracts the service name from each line stored in the variable "line" by printing the third field using the "awk" command.

port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

This command extracts the port number from each line stored in the variable "line" by first using ":" as the field separator to isolate the port section, then using "/" as the field separator to extract the port number.

hydra -L $file_name_path -P $file_pass_path $service://$ip:$port |grep -wi "login:" -B1 |tee -a $location/.../"Hydra $ip"

This command utilizes Hydra to perform a brute force attack, attempting login combinations from specified username and password lists. The "-L" flag specifies the username list file, "-P" specifies the password list file, and "$service://$ip:$port" specifies the target service, IP address, and port. The "grep" command filters for lines containing "login:" along with the preceding line. Finally, "tee -a" appends the output to a file named after the IP address in the "WeakPassRes" directory.

This Function will be responsible for conducting Brute force attack using hydra using a passwords and usernames from the user, then printing reports.

```
function UserWordsLog () # This Fucntion will be responsible for conducting Brute force attack using hydra using a passwords and usernames from the user, then printing reports.
{
    clear
    echo
    cd $location/$DirName
    mkdir Login_File > /dev/null 2>&1
    cd Login_File
    echo "[»] Please Enter The Usernames You Wish to Use:"
    sleep 1
    echo "[!] Make sure to press 'Enter' after each username. (Press CTRL+D when finished)"
    sleep 1
    echo

    echo "Your Chosen Usernames: "
    echo
    cat > namelist.txt
    echo
    echo
    echo "[»] Please Enter The Passwords You Wish to Use:"
    sleep 1
    echo "[!] Make sure to press 'Enter' after each Password. (Press CTRL+D when finished)"
    sleep 1
    echo
    echo "Your Chosen Passwords: "
    echo
    cat > password.lst
    echo
    echo
    echo "[√] Thank you! Your data input has been successfully recorded! "
    sleep 1
    echo
    echo "[*] Please wait while we initiate the Brute Force attack..."
    sleep 1
    echo "[!] Any results that have been captured will be displayed on the screen!"
    echo
    cd ../NmapScanRes
    cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line;
    do

        ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)
        service=$(echo "$line" | awk '{print $3}')
        port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

        hydra -L $location/$DirName/Login_File/namelist.txt -P $location/$DirName/Login_File/password.lst  $service://$ip:$port 2>/dev/null |grep -wi "login:" -B1 |tee -a $location/$DirName/WeakPassRes/"Hydra $i
        echo

    done
}
```

cat > namelist.txt/password.lst

This command opens a text editor for interactive input, allowing the user to type and create content in a file named "namelist.txt" or password.lst. Once the user is done typing, they can save the file by pressing Ctrl + D.

ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)

This command extracts the IP address from each line stored in the variable "line".
"echo "$line"" prints the current line, "awk '{print $1}'" extracts the first field (presumably the IP address) from the line. "cut -d ':' -f 1" separates the IP address from any port number by using ":" as a delimiter and selects the first field before ":".
The resulting IP address is stored in the variable "ip".

service=$(echo "$line" | awk '{print $3}')

This command extracts the service name from each line stored in the variable "line" by printing the third field using the "awk" command.

port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

This command extracts the port number from each line stored in the variable "line" by first using ":" as the field separator to isolate the port section, then using "/" as the field separator to extract the port number.

hydra -L $location/../namelist.txt -P $location/../password.lst $service://$ip:$port |grep -wi "login:" -B1 |tee -a $location/.../"Hydra $ip"

This command utilizes Hydra to perform a brute force attack, attempting login combinations from specified username and password lists. The "-L" flag specifies the username list file, "-P" specifies the password list file, and "$service://$ip:$port" specifies the target service, IP address, and port. The "grep" command filters for lines containing "login:" along with the preceding line. Finally, "tee -a" appends the output to a file named after the IP address in the "WeakPassRes" directory.

This function will be responsible for determining the Password and Username files to be utilized in the brute force attack.
To initiate the Brute Force Attack, Usernames and Passwords files are essential.
The user is presented with a menu offering three options: Auto, Path, Manual.

```
function LoginChoice () # This function will be responsible for determining the Password and Username files to be utilized in the brute force attack. To initia
{
    while true
    do
        clear
        echo
        echo "${bold}[*] To Initiate the Brute Force Attack, Usernames and Passwords Files are Essential."
        echo "[»] Please Choose the Way You Wish to Provide Those Files, According to the Following Options:"
        echo
        echo "${yellow} A) Auto - Automatically generate default usernames and passwords files that contains most known usernames and passwords."
        echo " P) Path - Enter paths to your own files that files containing usernames and passwords that you wish to use in the brute force."
        echo " M) Manual- Provides the option to manually input few usernames and passwords on the screen for utilization in the brute force attack. ${normal}"
        echo
        echo "${bold}${red}    +-----------------------------------------------------+"
        echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
        echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
        echo "    +-----------------------------------------------------+    ${normal}"
        echo

        echo "${bold}    ┌─[Enter Your Choice]"
        read -p "    └─────> " UserChoice
        Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

        case $Choice in

            a|auto)
                echo "            └─────>${purple} You Have Chosen to Use Automatically Generate Files! ${normal}"
                sleep 4
                AutoFiles
                break
            ;;
            p|path)
                echo "            └─────>${purple} You Have Chosen the Option to Enter a Path to Your Files!    ${normal}"
                sleep 4
                UserFiles
                break
            ;;
```

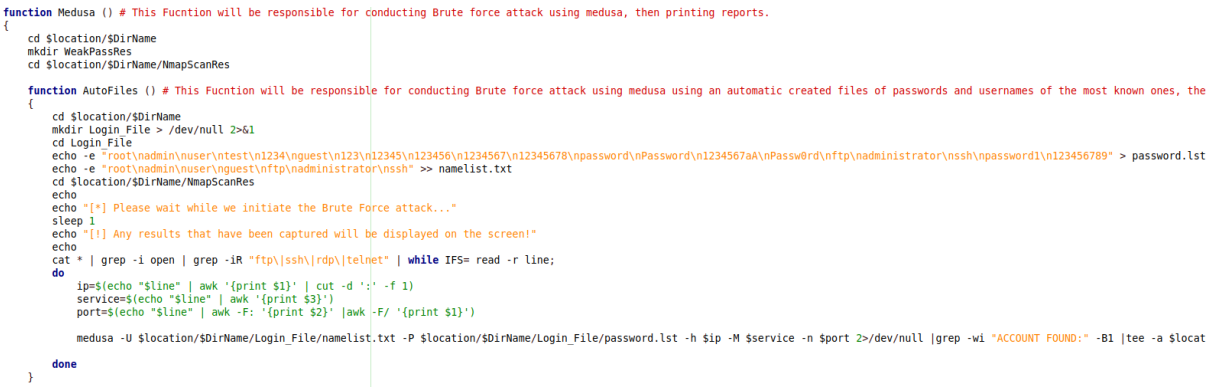
Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

This Function will be responsible for conducting Brute force attack using medusa, then printing reports.



```
function Medusa () # This Fucntion will be responsible for conducting Brute force attack using medusa, then printing reports.
{
    cd $location/$DirName
    mkdir WeakPassRes
    cd $location/$DirName/NmapScanRes

    function AutoFiles () # This Fucntion will be responsible for conducting Brute force attack using medusa using an automatic created files of passwords and usernames of the most known ones, the
    {
        cd $location/$DirName
        mkdir Login_File > /dev/null 2>&1
        cd Login_File
        echo -e "root\nadmin\nuser\ntest\n1234\nguest\n123\n12345\n123456\n1234567\n12345678\npassword\nPassword\n1234567aA\nPassw0rd\nftp\nadministrator\nssh\npassword1\n123456789" > password.lst
        echo -e "root\nadmin\nuser\nguest\nftp\nadministrator\nssh" >> namelist.txt
        cd $location/$DirName/NmapScanRes
        echo
        echo "[*] Please wait while we initiate the Brute Force attack..."
        sleep 1
        echo "[!] Any results that have been captured will be displayed on the screen!"
        echo
        cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line;
        do
            ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)
            service=$(echo "$line" | awk '{print $3}')
            port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

            medusa -U $location/$DirName/Login_File/namelist.txt -P $location/$DirName/Login_File/password.lst -h $ip -M $service -n $port 2>/dev/null |grep -wi "ACCOUNT FOUND:" -B1 |tee -a $locat

        done
    }
```

echo -e "root\nadmin\nuser\ntest\n1234\...\....\....\...\npassword1\n123456789" > password.lst

This command creates a file named "password.lst" and populates it with a list of common passwords separated by newline characters.
The "-e" flag enables interpretation of backslash escapes, allowing for the inclusion of newline characters ("\n") to separate each password entry.

echo -e "root\nadmin\nuser\nguest\nftp\nadministrator\nssh" >> namelist.txt

This command appends a list of usernames to the file "namelist.txt".
Each username is separated by a newline character ("\n"), specified by the "-e" flag with the "echo" command.
The ">>" operator appends the output to the end of the existing file "namelist.txt".

cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line

This command pipes the output of "cat *" (which concatenates the contents of all files in the current directory) to two "grep" commands.
The first "grep" filters for lines containing the case-insensitive word "open".
The second "grep" command further filters the output, searching for lines containing any of the specified protocols ("ftp", "ssh", "rdp", "telnet").
The results are then processed line by line in a "while" loop, where each line is stored in the variable "line".

ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)

This command extracts the IP address from each line stored in the variable "line".
"echo "$line"" prints the current line, "awk '{print $1}'" extracts the first field (presumably the IP address) from the line. "cut -d ':' -f 1" separates the IP address from any port number by using ":" as a delimiter and selects the first field before ":".
The resulting IP address is stored in the variable "ip".

service=$(echo "$line" | awk '{print $3}')

This command extracts the service name from each line stored in the variable "line" by printing the third field using the "awk" command.

port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

This command extracts the port number from each line stored in the variable "line" by first using ":" as the field separator to isolate the port section, then using "/" as the field separator to extract the port number.

medusa -U $location/…/namelist.txt -P $location/…/password.lst -h $ip -M $service -n $port |grep -wi "ACCOUNT FOUND:" -B1 |tee -a $location/…/WeakPassRes/"Medusa $ip"

This command executes a password cracking attempt using Medusa, targeting the IP address "$ip" with a specified username list file ("-U") and password list file ("-P"). The "-h" flag specifies the host to attack, and "-M" specifies the service. The "-n" flag specifies the port number. The "grep" command filters and prints lines containing "ACCOUNT FOUND:" along with the preceding line. Finally, "tee -a" appends the output to a file named after the IP address in the "WeakPassRes" directory.

This Function will be responsible for conducting Brute force attack using medusa using a file of passwords and usernames from the user, then printing reports.

```
function UserFiles () # This Fucntion will be responsible for conducting Brute force attack using medusa using a files of passwords and usernames from the user, then pr
{
    clear
    echo
    while true
    do
        read -p "[»] Please Enter a Path to a File That Contains Usernames That You Wish to Use: " file_name_path
        if [ -f "$file_name_path" ]
        then
            sleep 1
            echo
            echo
            echo "[✓] File is Exists and Valid to Use. Progressing... "
            echo
            echo
            break
        else
            echo
            echo
            sleep 2
            echo "[!] File Not Found, Please Try Again... "
            echo
            echo
            sleep 2
        fi
    done

    while true
    do
        read -p "[»] Please Enter a Path to a File That Contains Passwords That You Wish to Use: " file_pass_path
        if [ -f "$file_pass_path" ]
        then
            sleep 1
            echo




            echo
            echo "[✓] File is Exists and Valid to Use. Progressing... "
            echo
            echo
            break
        else
            echo
            echo
            sleep 2
            echo "[!] File Not Found, Please Try Again... "
            echo
            echo
            sleep 2
        fi
    done
    echo
    echo
    echo "[√] Thank you! Your data input has been successfully recorded! "
    sleep 1
    echo
    echo "[*] Please wait while we initiate the Brute Force attack..."
    sleep 1
    echo "[!] Any results that have been captured will be displayed on the screen!"
    echo
    cd NmapScanRes
    cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line;
    do

        ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)
        service=$(echo "$line" | awk '{print $3}')
        port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

        medusa -U $file_name_path -P $file_pass_path -h $ip -M $service -n $port 2>/dev/null |grep -wi "ACCOUNT FOUND:" -B1 |tee -a $location/$DirName/WeakPassRes/"Medusa $ip"


    done
}
```

if [-f "$file_name_path"]

This conditional statement checks if the file specified by the variable "$file_name_path" exists using the "-f" flag. If the file exists, the condition evaluates to true.

if [-f "$file_pass_path"]

This conditional statement checks if the file specified by the variable "$file_pass_path" exists using the "-f" flag. If the file exists, the condition evaluates to true.

cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line

This command pipes the output of "cat *" (which concatenates the contents of all files in the current directory) to two "grep" commands.
The first "grep" filters for lines containing the case-insensitive word "open".
The second "grep" command further filters the output, searching for lines containing any of the specified protocols ("ftp", "ssh", "rdp", "telnet").
The results are then processed line by line in a "while" loop, where each line is stored in the variable "line".

ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)

This command extracts the IP address from each line stored in the variable "line".
"echo "$line"" prints the current line, "awk '{print $1}'" extracts the first field (presumably the IP address) from the line. "cut -d ':' -f 1" separates the IP address from any port number by using ":" as a delimiter and selects the first field before ":".
The resulting IP address is stored in the variable "ip".

service=$(echo "$line" | awk '{print $3}')

This command extracts the service name from each line stored in the variable "line" by printing the third field using the "awk" command.

port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

This command extracts the port number from each line stored in the variable "line" by first using ":" as the field separator to isolate the port section, then using "/" as the field separator to extract the port number.

medusa -U $file_name_path -P $file_pass_path -h $ip -M $service -n $port 2>/dev/null |grep -wi "ACCOUNT FOUND:" -B1 |tee -a $location/$DirName/WeakPassRes/"Medusa $ip"
This command employs Medusa to perform a password cracking attempt, using a specified username list file ("-U") and password list file ("-P"). The target host is specified with the "-h" flag, while the service and port are provided using the "-M" and "-n" flags, respectively. The "2>/dev/null" redirects any error messages to null, suppressing them. The "grep" command filters and displays lines containing "ACCOUNT FOUND:" along with the preceding line. Finally, "tee -a" appends the output to a file named after the IP address in the "WeakPassRes" directory.

This Function will be responsible for conducting Brute force attack using medusa using a passwords and usernames from the user, then printing reports.

```
function UserWordsLog () # This Fucntion will be responsible for conducting Brute force attack using medusa using a passwords and usernames from the user, then printing reports.
{
    clear
    echo
    cd $location/$DirName
    mkdir Login_File > /dev/null 2>&1
    cd Login_File
    echo "[»] Please Enter The Usernames You Wish to Use:"
    sleep 1
    echo "[!] Make sure to press 'Enter' after each username. (Press CTRL+D when finished)"
    sleep 1

    echo
    echo "Your Chosen Usernames: "
    echo
    cat > namelist.txt
    echo
    echo
    echo "[»] Please Enter The Passwords You Wish to Use:"
    sleep 1
    echo "[!] Make sure to press 'Enter' after each Password. (Press CTRL+D when finished)"
    sleep 1
    echo
    echo "Your Chosen Passwords: "
    echo
    cat > password.lst
    echo
    echo
    echo "[√] Thank you! Your data input has been successfully recorded! "
    sleep 1
    echo
    echo "[*] Please wait while we initiate the Brute Force attack..."
    sleep 1
    echo "[!] Any results that have been captured will be displayed on the screen!"
    echo
    cd ../NmapScanRes
    cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line;
    do

        ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)
        service=$(echo "$line" | awk '{print $3}')
        port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

        medusa -U $location/$DirName/Login_File/namelist.txt -P $location/$DirName/Login_File/password.lst -h $ip -M $service -n $port 2>/dev/null |grep -wi "ACCOUNT FOUND:" -B1 |tee -a $location/$DirName

    done
}
```

cat > namelist.txt/password.lst

This command opens a text editor for interactive input, allowing the user to type and create content in a file named "namelist.txt" or password.lst. Once the user is done typing, they can save the file by pressing Ctrl + D.

cat * | grep -i open | grep -iR "ftp\|ssh\|rdp\|telnet" | while IFS= read -r line

This command pipes the output of "cat *" (which concatenates the contents of all files in the current directory) to two "grep" commands.
The first "grep" filters for lines containing the case-insensitive word "open".
The second "grep" command further filters the output, searching for lines containing any of the specified protocols ("ftp", "ssh", "rdp", "telnet").
The results are then processed line by line in a "while" loop, where each line is stored in the variable "line".

ip=$(echo "$line" | awk '{print $1}' | cut -d ':' -f 1)

This command extracts the IP address from each line stored in the variable "line".
"echo "$line"" prints the current line, "awk '{print $1}'" extracts the first field (presumably the IP address) from the line. "cut -d ':' -f 1" separates the IP address from any port number by using ":" as a delimiter and selects the first field before ":".
The resulting IP address is stored in the variable "ip".

service=$(echo "$line" | awk '{print $3}')

This command extracts the service name from each line stored in the variable "line" by printing the third field using the "awk" command.

port=$(echo "$line" | awk -F: '{print $2}' |awk -F/ '{print $1}')

This command extracts the port number from each line stored in the variable "line" by first using ":" as the field separator to isolate the port section, then using "/" as the field separator to extract the port number.

medusa -U $location/…/namelist.txt -P $location/…/password.lst -h $ip -M $service -n $port |grep -wi "ACCOUNT FOUND:" -B1 |tee -a $location/…/WeakPassRes/"Medusa $ip"

This command executes a password cracking attempt using Medusa, targeting the IP address "$ip" with a specified username list file ("-U") and password list file ("-P"). The "-h" flag specifies the host to attack, and "-M" specifies the service. The "-n" flag specifies the port number. The "grep" command filters and prints lines containing "ACCOUNT FOUND:" along with the preceding line. Finally, "tee -a" appends the output to a file named after the IP address in the "WeakPassRes" directory.

This function will be responsible for determining the Password and Username files to be utilized in the brute force attack.
To initiate the Brute Force Attack, Usernames and Passwords files are essential.
The user is presented with a menu offering three options: Auto, Path, Manual.

```
function LoginChoice () # This function will be responsible for determining the Password and Username files to be utilized
{
    while true
    do
        clear
        echo
        echo "${bold}[*] To Initiate the Brute Force Attack, Usernames and Passwords Files are Essential."
        echo "[»] Please Choose the Way You Wish to Provide Those Files, According to the Following Options:"
        echo
        echo "${yellow} A) Auto - Automatically generate default usernames and passwords files that contains most known use
        echo " P) Path - Enter paths to your own files that files containing usernames and passwords that you wish to use
        echo " M) Manual- Provides the option to manually input few usernames and passwords on the screen for utilization
        echo
        echo "${bold}${red}    +-----------------------------------------------------+"
        echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
        echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
        echo "    +-----------------------------------------------------+    ${normal}"
        echo

        echo "${bold}    ┌─[Enter Your Choice]"
        read -p "    └──────> " UserChoice
        Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

        case $Choice in

            a|auto)
                echo "            └──────>${purple} You Have Chosen to Use Automatically Generate Files! ${normal}"
                sleep 4
                AutoFiles
                break
            ;;
            p|path)
                echo "            └──────>${purple} You Have Chosen the Option to Enter a Path to Your Files!    ${normal}"
                sleep 4
                UserFiles
                break

            ;;
            m|manual)
                echo "            └──────>${purple} You Have Chosen the Manual Option to Manually Input Login Details!   ${normal}"
                sleep 4
                UserWordsLog
                break
            ;;
            99)
                echo "            └──────> ${red}Going Back to Previous Menu${normal}${bold} ┐"
                echo "                                                 <┘    ${normal}"
                sleep 3
                NmapMasscan
                break
            ;;
            ..|exit)
                echo "            └──────>${red} Exiting the framework... ${normal}"
                sleep 2
                exit
            ;;
            *)
                echo "            └──────>${green} Invalid choice! Please try again... ${normal}"
                sleep 3
            ;;

        esac
    done
}
LoginChoice
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

This Function will be responsible for conducting Nse Brute analysis, then printing reports.

```
function Nse () # This Fucntion will be responsible for conducting Nse Brute analysis, then printing reports.
{
        echo
        echo
        echo "[*] Please wait while we initiate the NSE (Nmap Scripting Engine) Brute Force attack..."
        sleep 1
        echo
        echo "[*] After the Brute Force attempts are completed, a report will be generated and saved in your chosen directory. "
        echo "[!] Any results that have been captured will be displayed on the screen!"
        echo
        cd $location/$DirName
        mkdir WeakPassRes 2>/dev/null
        cd $location/$DirName
        mkdir NseReport 2>/dev/null
        cd NseReport
        mkdir NseBrute 2>/dev/null
        cd $location/$DirName/NmapScanRes
        for ip in $(cat * |grep -iR open |awk -F: '{print $1}'|sort |uniq )
        do
            sudo nmap -sV -T4 $ip --script=brute >>  $location/$DirName/NseReport/NseBrute/"NseBrute $ip" 2>/dev/null
            if cat $location/$DirName/NseReport/NseBrute/"NseBrute $ip" |grep -i "Valid credentials" >/dev/null;
            then
                echo "[!] Great news! We've found some valid credentials for the ip $ip: "
                echo
                sleep 2
                cat $location/$DirName/NseReport/NseBrute/"NseBrute $ip" |grep -i "Valid credentials"
                echo "[!] Great news! We've found some valid credentials for the ip $ip: " > $location/$DirName/WeakPassRes/"Nse $ip"
                echo " " >> $location/$DirName/WeakPassRes/"Nse $ip"
                cat $location/$DirName/NseReport/NseBrute/"NseBrute $ip" |grep -i "Valid credentials" >> $location/$DirName/WeakPassRes/"Nse $ip"
            else
                echo "[!] Uh-oh! We didn't find any valid credentials for the ip $ip"
                echo "[!] Uh-oh! We didn't find any valid credentials for the ip $ip" > $location/$DirName/WeakPassRes/"Nse $ip"
            fi
        done
}
```

for ip in $(cat * |grep -iR open |awk -F: '{print $1}'|sort |uniq)

This command iterates through each unique IP address extracted from lines containing the word "open" across all files in the current directory. It concatenates the contents of all files using "cat *", then searches recursively for lines containing "open" with "grep -iR open". Using "awk -F: '{print $1}'", it extracts the IP address portion before the colon ":" delimiter. The IP addresses are sorted alphabetically with "sort" and duplicates are filtered out using "uniq". Each unique IP address is then assigned to the variable "ip" for further processing within the loop.

sudo nmap -sV -T4 $ip --script=brute >> $location/…/NseBrute/"NseBrute $ip"

This command utilizes nmap to conduct a version detection scan ("-sV") with aggressive timing template ("-T4") on the specified IP address "$ip". Additionally, it executes brute force scripts using the "--script=brute" option to attempt login credentials. The results are appended to a file named after the IP address in the "NseBrute" directory within the specified location and directory.

if cat $location/$DirName/NseReport/NseBrute/"NseBrute $ip" |grep -i "Valid credentials"

This conditional statement checks if the output file generated by nmap contains the phrase "Valid credentials" using the "grep" command. If the phrase is found, indicating successful login attempts, the condition evaluates to true.

This function will be responsible for determining which cracking tool to use in the brute force attack aimed at cracking weak passwords that might be found within the weaknesses identified in the scanning results: Hydra, Medusa, NSE Brute.

```
function BFChoice () # This function will be responsible for determining which cracking tool to use in the
{
    while true
    do
        clear
        echo
        echo "${bold}[✓] After successfully completing the scan, we'll initiate a brute force attack to unc
        echo "[»] What type of cracking tool do you prefer for the brute force attack?"
        echo
        echo "${yellow} H) Hydra - A versatile and powerful brute force tool known for its ability to crack
        echo " M) Medusa - A command-line tool designed for fast and parallelized password cracking, suppor
        echo " N) NSE Brute - Nmap Scripting Engine tool used for brute force attacks to discover weak pass
        echo
        echo "${bold}${red}    +-----------------------------------------------------+"
        echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
        echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
        echo "    +-----------------------------------------------------+ ${normal}"
        echo

        echo "${bold}     ┌─[Enter Your Choice]"
        read -p  "     └─────> " UserChoice
        Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

        case $Choice in

            h|hydra)
                echo "          └─────>${purple} You Have Chosen to use Hydra Cracking Tool!  ${normal}"
                sleep 3
                Hydra
                break
            ;;
            m|medusa)
                echo "          └─────>${purple} You Have Chosen to use Medusa Cracking Tool! ${normal}"
                sleep 3
                Medusa
                break

            ;;
            nse|n|"nse brute")
                echo "          └─────>${purple} You Have Chosen to use NSE Brute Cracking Tool!  ${normal}"
                sleep 3
                Nse
                break
            ;;
            99)
                echo "          └─────>${red} Going Back to Previous Menu${normal}${bold} ─┐"
                echo "                                                    <─┘ ${normal}"
                sleep 3
                NmapMasscan
                break
            ;;
            ..|exit)
                echo "          └─────>${red} Exiting the framework... ${normal}"
                sleep 2
                exit
            ;;
            *)
                echo
                echo "          └─────>${red} Invalid choice! Please try again... ${normal}"
                sleep 3
            ;;

        esac
    done

}
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

This Function will be responsible for conducting masscan scan.

The subsequent functions and commands within the MasscanScan function are essentially identical to the NmapScan function, differing only in the specific commands used for scanning with Masscan, which will be elaborated upon in this section !

The following command employs Masscan to scan all ports (" -p- ") on the specified IP address "$IP" at a rate of 1000 packets per second (" --rate 1000 ").
The results are saved in the "MasscanScanRes" directory in a format suitable for further parsing (" -oG ").

```
sudo masscan -p- --rate 1000 $IP -oG MasscanScanRes/$IP
```

The following command executes Masscan with elevated privileges to perform a port scan on all ports ("-p-") of the specified IP address "$IP".
The results are saved in the "MasscanScanRes" directory in a format suitable for further parsing ("-oG").

```
sudo masscan -p- $IP -oG MasscanScanRes/$IP
```

The following command utilizes Masscan with elevated privileges to perform a UDP port scan ("-pU:0-65535") on all ports ranging from 0 to 65535 of the specified IP address "$IP". It sets the scan rate to 10000 packets per second ("--rate 10000") and saves the results in the "MasscanScanRes" directory in a format suitable for further parsing ("-oG").

```
sudo masscan -pU:0-65535 --rate 10000 $IP -oG MasscanScanRes/$IP
```

The following command executes Masscan with elevated privileges to perform a UDP port scan ("-pU:0-65535") on all ports ranging from 0 to 65535 of the specified IP address "$IP". The scan results are saved in the "MasscanScanRes" directory in a format suitable for further parsing ("-oG").

```
sudo masscan -pU:0-65535 $IP -oG MasscanScanRes/$IP
```

The function will be responsible for determining what scanning tool would be used, each with its own advantages depending on the specific requirements. It will ask the user what type of scanning tool he want to execute Nmap/Masscan:

```
function NmapMasscan () # The function will be responsible for determining what scanning tool would be us
{
    while true
    do
        clear
        echo
        echo "${bold}[»] What kind of scanning tool would you like to use: Nmap or Masscan?"
        echo
        echo "${yellow} N) Nmap - Known for its reliability, Nmap offers thorough and detailed scanning c
        echo " M) Masscan - Known for its high-speed scanning capabilities, it may sacrifice some reliabi
        echo
        echo "${bold}${red}    +-----------------------------------------------------------+"
        echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
        echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
        echo "    +-----------------------------------------------------------+  ${normal}"
        echo

        echo "${bold}    ┌─[Enter Your Choice]"
        read -p "    └─────► " UserChoice
        Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

        case $Choice in

            n|nmap)
                echo "            └─────►${purple} You Have Chosen to Scan with Nmap Tool!  ${normal}"
                sleep 3
                NmapScan
                break
            ;;
            m|masscan)
                echo "            └─────►${purple} You Have Chosen to Scan with Masscan Tool!  ${normal}"
                sleep 3
                MasscanScan
                break
            ;;
            99)

                echo "            └─────►${red} Going Back to Previous Menu ${normal}${bold}─┐"
                echo "                                                    ◄─┘ ${normal}"
                sleep 3
                CaseChoice
                break
            ;;
            ..|exit)
                echo "            └─────►${red} Exiting the framework... ${normal}"
                sleep 2
                exit
            ;;
            *)
                echo "            └─────►${green} Invalid choice! Please try again...  ${normal}"
                sleep 3
            ;;

        esac
    done

}
NmapMasscan
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

This Function will be responsible for utilizing the Nmap Scripting Engine (NSE) for vulnerability analysis, including Searchsploit and vulnerability identification.

```
function Full () # This Fucntion will be responsible for utilizes the Nmap Scripting Engine (NSE) for vulnerability analysis, including Searchspl
{
    cd $location/$DirName
    mkdir NseReport  2>/dev/null

    function NseVuln () # This Fucntion will be responsible for scanning the target network for vulnerability analysis, then printing reports.
    {
        echo
        echo
        echo "[*] Please wait while we initiate the NSE (Nmap Scripting Engine) Vulnerability Analysis..."
        sleep 1
        echo
        echo "[*] After the Vulnerability Analysis are completed, a report will be generated and saved in your chosen directory. "
        cd $location/$DirName/NseReport
        mkdir NseVuln 2>/dev/null
        cd $location/$DirName/*ScanRes
        for ip in $(cat * |grep -iR open |awk -F: '{print $1}'|sort |uniq )
        do
            nmap -sV --open -T4 --script vuln $ip -oX $location/$DirName/NseReport/NseVuln/NseVulnRes.xml > /dev/null 2>&1
            xsltproc $location/$DirName/NseReport/NseVuln/NseVulnRes.xml -o $location/$DirName/NseReport/NseVuln/NseVulnRes.html > /dev/null 2>&1
        done
        echo
        echo
        echo "${bold}${green}[!] The Vulnerability Analysis has been successfully completed, a report of the scan has been successfully generated
        echo
    }
    NseVuln
```

for IP in $(cat IPRangeFile |awk '{print $(NF)}'| grep ^[0-9])

This "for" loop iterates over each IP address listed in the "IPRangeFile" file, extracting the last field of each line using "awk '{print $(NF)}'", which represents the IP address.
The "grep ^[0-9]" command filters out any lines that do not start with a digit, ensuring that only valid IP addresses are processed within the loop.

nmap -sV --open -T4 --script vuln $ip -oX $location/../NseReport/NseVuln/NseVulnRes.xml

This command utilizes nmap to conduct a version detection scan ("-sV") on the specified IP address "$ip", targeting only open ports ("--open") with aggressive timing template ("-T4").
Additionally, it executes vulnerability scripts using the "--script vuln" option to identify potential vulnerabilities.
The results are saved in XML format ("-oX") in the "NseVulnRes.xml" file located in the "NseVuln" directory within the specified location and directory.

xsltproc $location/…/NseVulnRes.xml -o $location/…/NseVulnRes.html

This command utilizes xsltproc to transform the XML file "NseVulnRes.xml" located in the "NseVuln" directory within the specified location and directory into an HTML format.
The resulting HTML file is saved as "NseVulnRes.html" in the same directory.

This Function will be responsible for conducting searchsploit analysis, then printing reports.

```
function Searchsploit () # This Fucntion will be responsible for conducting searchsploit analysis, then printing reports.
{
    echo
    echo "[*] Please wait as we utilize the generated files to conduct a vulnerability analysis using Searchsploit..."
    sleep 1
    echo
    echo "[*] After the Searchsploit Analysis is completed, a report will be generated and saved in your chosen directory. "
    cd $location/$DirName/NseReport
    mkdir Searchsploit 2>/dev/null
    cd $location/$DirName/NseReport/NseVuln
    searchsploit --exclude="Privilege Escalation" --disable-colour --nmap NseVulnRes.xml > $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt
    echo
    echo
    echo "${bold}${green}[!] The Searchsploit Analysis has been successfully completed, a report of the scan has been successfully generated! ${normal}"
    echo
    rm -rf $location/$DirName/NseReport/NseVuln/NseVulnRes.xml
    sleep 5

}
Searchsploit
```

searchsploit --exclude="Privilege Escalation" --disable-colour --nmap NseVulnRes.xml > $location/…/Searchsploit/SearchsploitRes.txt

This command uses searchsploit to search for exploits based on the results of a nmap scan stored in the XML file "NseVulnRes.xml".
The "--exclude" flag is used to exclude exploits related to "Privilege Escalation".
The "--disable-colour" flag disables colorized output, and the "--nmap" flag specifies that the input is a nmap XML file.
The results are saved in a text file named "SearchsploitRes.txt" in the "Searchsploit" directory within the specified location and directory.

rm -rf $location/$DirName/NseReport/NseVuln/NseVulnRes.xml

This command removes the XML file "NseVulnRes.xml" located in the "NseVuln" directory within the specified location and directory.
The "-rf" flags force removal of the file and its contents recursively without prompting for confirmation.

This function will display a menu to the user after completing the entire scanning process, allowing them to inspect all the files created during the scanning process. The options are: OpenIPs, IPRange, WeakCred, LogFile, NSE.

```
function ResualtMenu () # This function will display a menu to the user after completing the entire scanning process, allowing them to inspect all the files create
{
    cat $location/$DirName/IPRangeFile |awk '{print $(NF)}'|grep ^[0-9] > $location/$DirName/IPRangeScan
    cd $location/$DirName
    rm -rf IPRangeFile 2>/dev/null
    cd $location/$DirName/*ScanRes > /dev/null 2>&1
    mkdir FoundIP_Res 2>/dev/null
    for ValidIP in $(cat * |grep -iR open |awk -F: '{print $1}'|sort|uniq)
    do
        cp $ValidIP $location/$DirName/*ScanRes/FoundIP_Res > /dev/null 2>&1
    done

    function LogFile () # This function will be responsible for creating the Log file containing most of the information that was created.
    {
        cd $location/$DirName
        echo "[»] The following data represents the results of the scan performed! "  >> LogFile
        echo "    Please review the results and use them according to your requirements..." >> LogFile
        echo " " >> LogFile
        echo "[✓] Total count of IP addresses that have been scanned: $(cat $location/$DirName/IPRangeScan 2>/dev/null |wc -l)" >> LogFile
        echo " " >> LogFile
        echo "[»] The following IP addresses are those that were included in the scan: " >> LogFile
        echo " " >> LogFile
        echo "$(cat $location/$DirName/IPRangeScan  2>/dev/null | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total count of IP addresses that had an open ports: $(ls $location/$DirName/*ScanRes/FoundIP_Res/* 2>/dev/null |wc -l )" >> LogFile
        echo " " >> LogFile
        echo "[»] The following IP addresses were found to have open ports: " >> LogFile
        echo " " >> LogFile
        echo "$(ls $location/$DirName/*ScanRes/FoundIP_Res  2>/dev/null | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total count of weak credentials that were revealed during the scan: $(cat $location/$DirName/WeakPassRes/* |grep -Ev "Great news|\Uh-oh" |wc -l)"
        echo " " >> LogFile
        echo "[»] The following are the weak credentials that were discovered:" >> LogFile
        echo " " >> LogFile
        echo "$(cat $location/$DirName/WeakPassRes/* |grep -Ev "Great news|\Uh-oh"  2>/dev/null | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total Count of Vulnerabilities Detected for VSFTPD Service by 'Searchsploit': $(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt
        echo " " >> LogFile
        echo " " >> LogFile
        echo "[»] The following vulnerabilities have been detected for the VSFTPD service:"  >> LogFile
        echo " "  >> LogFile
        echo "$(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt   2>/dev/null  | grep -i vsftpd | sort | uniq | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total Count of Vulnerabilities Detected for OpenSSH Service by 'Searchsploit': $(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt
        echo " " >> LogFile
        echo "[»] The following vulnerabilities have been detected for the OpenSSH service:"  >> LogFile
        echo " " >> LogFile
        echo "$(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt   2>/dev/null  | grep -i OpenSSH | sort | uniq | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total Count of Vulnerabilities Detected for Telnet Service by 'Searchsploit': $(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt
        echo " " >> LogFile
        echo "[»] The following vulnerabilities have been detected for the Telnet service:"  >> LogFile
        echo " " >> LogFile
        echo "$(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt   2>/dev/null  | grep -i Telnet  | sort | uniq | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total Count of Vulnerabilities Detected for ISC BIND Service by 'Searchsploit': $(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.tx
        echo " " >> LogFile
        echo "[»] The following vulnerabilities have been detected for the ISC BIND service:"  >> LogFile
        echo " "  >> LogFile
        echo "$(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt   2>/dev/null  | grep -i ISC | sort | uniq | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total Count of Vulnerabilities Detected for Apache Service by 'Searchsploit': $(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt
        echo " " >> LogFile
        echo "[»] The following vulnerabilities have been detected for the Apache service:"  >> LogFile
        echo " " >> LogFile
        echo "$(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt   2>/dev/null  | grep -i Apache  | sort | uniq | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
        echo "[✓] Total Count of Vulnerabilities Detected for ProFTPd Service by 'Searchsploit': $(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt
        echo " " >> LogFile
        echo "[»] The following vulnerabilities have been detected for the ProFTPd service:"  >> LogFile
        echo " " >> LogFile
        echo "$(cat $location/$DirName/NseReport/Searchsploit/SearchsploitRes.txt   2>/dev/null  | grep -i ProFTPd  | sort | uniq | sed 's/^/    /') " >> LogFile
        echo " " >> LogFile
    }
    LogFile
```

cat $location/…/IPRangeFile |awk '{print $(NF)}'|grep ^[0-9] > $location/…IPRangeScan

This command reads the contents of the file "IPRangeFile" located in the specified directory "$location/$DirName". It then extracts the last field of each line using "awk '{print $(NF)}'", assuming it represents an IP address.
Next, it filters out lines starting with a digit (which likely indicates an IP address) using "grep ^[0-9]". Finally, it saves the filtered IP addresses to a new file named "IPRangeScan" in the same directory.

cd $location/$DirName/*ScanRes

This command changes the current directory to the directory containing files with names ending in "ScanRes", located within the specified directory "$location/$DirName".

for ValidIP in $(cat * |grep -iR open |awk -F: '{print $1}'| sort | uniq)

This command iterates through each unique IP address extracted from lines containing the word "open" across all files in the current directory. It concatenates the contents of all files using "cat *", then searches recursively for lines containing "open" with "grep -iR open". Using "awk -F: '{print $1}'", it extracts the IP address portion before the colon ":" delimiter. The IP addresses are then sorted alphabetically with "sort", and duplicates are filtered out using "uniq". Each unique IP address is assigned to the variable "ValidIP" for further processing within the loop.

cp $ValidIP $location/$DirName/*ScanRes/FoundIP_Res

This command copies the file or directory named "$ValidIP" to the "FoundIP_Res" directory within the directory matching the pattern "*ScanRes" located in the specified directory "$location/$DirName". If "$ValidIP" represents a file, it will be copied. If it represents a directory, its contents will be copied recursively.

This while statement will display a menu to the user after completing the entire scanning process, allowing them to inspect all the files created during the scanning process. The options are: OpenIPs, IPRange, WeakCred, LogFile, NSE.

```
while true
do
    clear
    echo
    echo "${bold}[»] From this menu, you can inspect your created files and directories. Please choose the data you want to display:"
    echo
    echo "${yellow} I) OpenIPs - Display the list of IP addresses that were discovered to have open ports during the scan."
    echo " R) IPRange - Display the list of IP addresses that were scanned during the process."
    echo " W) WeakCred - Display any weak credentials that were captured and detected during the scan."
    echo " L) LogFile - Display the created log file containing reports of the most important data."
    echo " N) NSE - Display the results of the NSE scans that were performed (if you choose to perform a scan).  ${normal}"
    echo
    echo "${bold}${red}     +-----------------------------------------------------------+"
    echo "     |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
    echo "     +-----------------------------------------------------------+  ${normal}"
    echo

    echo "${bold}     ┌─[Enter Your Choice]"
    read -p "     └─────► " UserChoice
    Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

    case $Choice in

        i|openips)
            echo "            └─────►${purple} You Have Chosen to view the OpenIPs file!    ${normal}"
            sleep 3
            cd $location/$DirName/*ScanRes/FoundIP_Res
            for foundip in $(cat * |grep -iR open |awk -F: '{print $1}'|sort|uniq)
            do
                echo
                echo
                echo "[»] The following presents the results for $foundip"
                echo
                cat $foundip
                echo
                echo
            done
        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press any key to continue...  ${normal}"
        read -n 1 -s -r
    ;;
    r|iprange)
        echo "            └─────►${purple} You Have Chosen to view the IPRange file! ${normal}"
        sleep 3
        firefox $location/$DirName/IPRangeScan &
        echo
        echo
        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press any key to continue...  ${normal}"
        read -n 1 -s -r
        pkill -f firefox >/dev/null 2>&1
    ;;
    w|weakcred)
        echo "            └─────►${purple} You Have Chosen to view the WeakCred file!  ${normal}"
        sleep 3
        firefox $location/$DirName/WeakPassRes/* &
        echo
        echo
        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press any key to continue... ${normal}"
        read -n 1 -s -r
        pkill -f firefox >/dev/null 2>&1
    ;;
    l|logfile)
        echo "            └─────►${purple} You Have Chosen to view the LogFile file!  ${normal}"
        sleep 3
        firefox $location/$DirName/LogFile &
        echo
        echo
        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press any key to continue... ${normal}"
        read -n 1 -s -r
        pkill -f firefox
    ::
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks.
Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

read -n 1 -s -r

This command reads a single character from standard input without displaying it on the terminal. It waits for user input, allowing the user to press any key, and then proceeds without displaying the entered character. The "-n 1" option specifies reading only one character, while the "-s" flag suppresses echoing of input characters to the terminal. Additionally, the "-r" flag disables interpretation of backslashes, ensuring that the input is treated as raw data.

firefox $location/$DirName/IPRangeScan &

This command launches the Firefox web browser and opens a new tab with the URL specified by the file located at "$location/$DirName/IPRangeScan". The ampersand ("&") at the end runs the command in the background, allowing you to continue using the terminal while Firefox opens.

pkill -f firefox

This command terminates all Firefox processes. It searches for processes containing the string "firefox" in their command line arguments ("-f") and terminates them.

Within the NSE option, a menu with three options will be displayed if the user selects NSE. This menu will allow the user to inspect the files created by NSE.

```
n|nse)
    echo "            └──────►${purple} You Have Chosen to view the NSE file!  ${normal}"
    sleep 3
    cd $location/$DirName >/dev/null 2>&1
    if [ -d "NseReport" ];
    then

        while true
        do
            clear
            cd NseReport >/dev/null 2>&1
            echo
            echo "${bold}[»] Please choose from the following options which NSE scan you want to display?"
            echo
            echo "${yellow} B) NseBrute - Displays the report generated by NSE Brute if a scan was performed.
            echo " V) NseVuln - Displays the report generated by NSE Vuln if a scan was performed."
            echo " S) Searchsploit - Displays the report generated by Searchsploit if a scan was performed.
            echo
            echo "${bold}${red}    +-----------------------------------------------------------+"
            echo "    |  99) To Return to the Previous Menu, Simply type '99'  |"
            echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
            echo "    +-----------------------------------------------------------+ ${normal}"
            echo

            echo "  ${bold}  ┌─[Enter Your Choice]"
            read -p  "    └──────► " UserChoice
            Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

            case $Choice in

                b|nsebrute)
                    echo "            └──────►${purple} You Have Chosen to view the NseBrute file!  ${normal}"
                    sleep 3
                    if [ -d "NseBrute" ];
                    then
                        firefox $location/$DirName/NseReport/NseBrute/* &
                        echo
                        echo
                        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press
                        read -n 1 -s -r
                        pkill -f firefox >/dev/null 2>&1
                    else
                        echo
                        echo
                        echo "${bold}${red}[!] Unfortunately, an NSE Brute scan hasn't been executed, and a repo
                        sleep 5
                    fi
                ;;
                v|nsevuln)
                    echo "            └──────►${purple} You Have Chosen to view the NseVuln file!  ${normal}"
                    sleep 3
                    if [ -d "NseVuln" ];
                    then
                        firefox $location/$DirName/NseReport/NseVuln/* &
                        echo
                        echo
                        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press
                        read -n 1 -s -r
                        pkill -f firefox >/dev/null 2>&1
                    else
                        echo
                        echo
                        echo "${bold}${red}[!] Unfortunately, an NSE Vuln scan hasn't been executed, and a repor
                        sleep 5
                    fi
                ;;
```

if [-d "NseVuln/ NseBrute"]

This conditional statement checks if a directory named "NseVuln/ NseBrute" exists in the current directory. If the directory exists, the condition evaluates to true.

firefox $location/$DirName/NseReport/NseVuln/* &

This command launches the Firefox web browser and opens each file within the "NseVuln" directory located at "$location/$DirName/NseReport/NseVuln" in separate tabs. The ampersand ("&") at the end runs the command in the background, allowing you to continue using the terminal while Firefox opens the files.

pkill -f firefox

This command terminates all Firefox processes. It searches for processes containing the string "firefox" in their command line arguments ("-f") and terminates them.

read -n 1 -s -r

This command reads a single character from standard input without displaying it on the terminal. It waits for user input, allowing the user to press any key, and then proceeds without displaying the entered character. The "-n 1" option specifies reading only one character, while the "-s" flag suppresses echoing of input characters to the terminal. Additionally, the "-r" flag disables interpretation of backslashes, ensuring that the input is treated as raw data.

```
s|searchsploit)
    echo "              └──────►${purple} You Have Chosen to view the Searchsploit file!  ${normal}"
    sleep 3
    if [ -d "Searchsploit" ];
    then
        firefox $location/$DirName/NseReport/Searchsploit/* &
        echo
        echo
        echo "${bold}${green}[»] Opening the file. After you have finished inspecting it, press any
        read -n 1 -s -r
        pkill -f firefox >/dev/null 2>&1
    else
        echo
        echo
        echo "${bold}${red}[!] Unfortunately, an Searchsploit scan hasn't been executed, and a repo
        sleep 5
    fi
;;
99)
    echo "              └──────►${red} Going Back to Previous Menu ${normal}${bold}┐"
    echo "                                                  ◄──┘ ${normal}"
    sleep 3
    break
::
```

firefox $location/$DirName/NseReport/Searchsploit/* &

This command opens each file within the "Searchsploit" directory located at "$location/$DirName/NseReport/Searchsploit" in separate tabs in the Firefox web browser. The ampersand ("&") at the end runs the command in the background, allowing you to continue using the terminal while Firefox opens the files.

read -n 1 -s -r

This command reads a single character from standard input without displaying it on the terminal. It waits for user input, allowing the user to press any key, and then proceeds without displaying the entered character. The "-n 1" option specifies reading only one character, while the "-s" flag suppresses echoing of input characters to the terminal. Additionally, the "-r" flag disables interpretation of backslashes, ensuring that the input is treated as raw data.

pkill -f firefox

This command terminates all Firefox processes. It searches for processes containing the string "firefox" in their command line arguments ("-f") and terminates them.

will prompt the user to choose whether they want to zip and compress the directory containing all the data or keep it as an uncompressed, normal directory. Yes/No:

```
..|exit)
    while true
    do
        clear
        echo
        echo "${bold}[»] Before exiting the framework, would you like to zip the created directory?   "
        echo
        echo "${yellow} Y) Yes -  Select 'Yes' if you wish to zip your created directory before exiting..."
        echo " N) No - Select 'No' if you wish to keep your created directory as is and unzipped.   ${normal}"
        echo
        echo "${bold}${red}    +-----------------------------------------------------------+"
        echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
        echo "    +-----------------------------------------------------------+ ${normal}"
        echo

        echo " ${bold}    ┌─[Enter Your Choice]"
        read -p "     └─────► " UserChoice
        Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

        case $Choice in

            y|yes)
                zip -r "$DirName.zip" "$DirName" >/dev/null 2>&1
                rm -rf "$DirName" >/dev/null 2>&1
                echo "              └─────► ${purple}The directory has been successfully zipped! ${normal}"
                echo
                sleep 2
                echo "          ${bold}                                  |"
                echo "         ${bold}                                   ▼ "
                echo "            ${bold}       ${red} Thank you for using the framework! Enjoy your explorati
                sleep 3
                exit
            ;;
            n|no)
                echo "              └─────►${red} Thank you for using the framework! Enjoy your exploration... $
                sleep 3
                exit
            ;;
            ''
            ..|exit)
                echo "              └─────►${red} Thank you for using the framework! Enjoy your exploration... ${normal}"
                sleep 2
                exit
            ;;
            *)
                echo "              └─────► ${green}Invalid choice! Please try again... ${normal}"
                sleep 3
            ;;

        esac
    done
;;
*)
    echo "              └─────►${green} Invalid choice! Please try again... ${normal}"
    sleep 3
;;

    esac
done


else
    echo
    echo "${bold}${red}[!] Unfortunately, an NSE scan hasn't been executed, and a report file doesn't exist. ${normal}"
    sleep 5
fi
;;
```

Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

In this command, the variable "Choice" is assigned the result of converting the content of the variable "$UserChoice" to lowercase using the "tr" command.
This ensures that the user's input is standardized to lowercase characters for consistent comparison and processing in the script.

case $choice in

This "case" statement checks the value of the variable "$choice" against various patterns or conditions specified in subsequent " ;; " blocks. Based on the value of "$choice", the script will execute the commands associated with the matching pattern.
In each choice that is available there is a function being called and if he enters invalid choice that doesn't exist the *) operate and loop until the user enter valid choice.

zip -r "$DirName.zip" "$DirName"

This command creates a zip archive named "$DirName.zip" containing all files and directories within the directory named "$DirName". The "-r" flag specifies that the compression should be recursive, including all files and subdirectories within "$DirName".

rm -rf "$DirName"

This command recursively removes the directory named "$DirName" and all its contents. The "-rf" flags force removal without prompting for confirmation and ensure that even write-protected files within the directory are removed.

This Last function will be responsible for determining the depth of the scan and will ask the user what kind of scan he want to perform Basic/Full:

```
function CaseChoice () # This Function will be responsible for determining the depth of the scan and will ask the user what k
{
    while true
    do
        clear
        echo
        echo "${bold}[»] Please Choose the Kind of Scan You Want to Preform:   "
        echo
        echo "${yellow} B) Basic - Scans the network for TCP and UDP, including the service version and weak passwords."
        echo " F) Full  - Scans the network include Nmap Scripting Engine (NSE), weak passwords, and vulnerability analysis.
        echo
        echo "${bold}${red}    +-----------------------------------------------------+"
        echo "    |  ..) To Exit The Framework, Simply type '..' or 'Exit' |"
        echo "    +-----------------------------------------------------+ ${normal}"
        echo

        echo " ${bold}    ┌─[Enter Your Choice]"
        read -p "    └─────> " UserChoice
        Choice=$(echo $UserChoice |tr '[:upper:]' '[:lower:]')

        case $Choice in

            b|basic)
                echo "            └─────> ${purple}You Have Chosen Basic Scan! ${normal}"
                sleep 3
                Basic
                ResualtMenu
                break
            ;;
            f|full)
                echo "            └─────> ${purple}You Have Chosen Full Scan!   ${normal}"
                sleep 3
                Basic
                Full
                ResualtMenu
                break
            ;;
            ..|exit)
                echo "             └─────> ${red}Exiting the framework...   ${normal}"
                sleep 2
                exit
            ;;
            *)
                echo "            └─────> ${green}Invalid choice! Please try again... ${normal}"
                sleep 3
            ;;

        esac
    done
}
CaseChoice
```

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!