

# PENETRATION TESTING

## **PROJECT: Vulner**

Student: David lim

Student code: s7

Python Fundamentals trainer: karwei

Class code: CFC020823

## Introduction to the Project

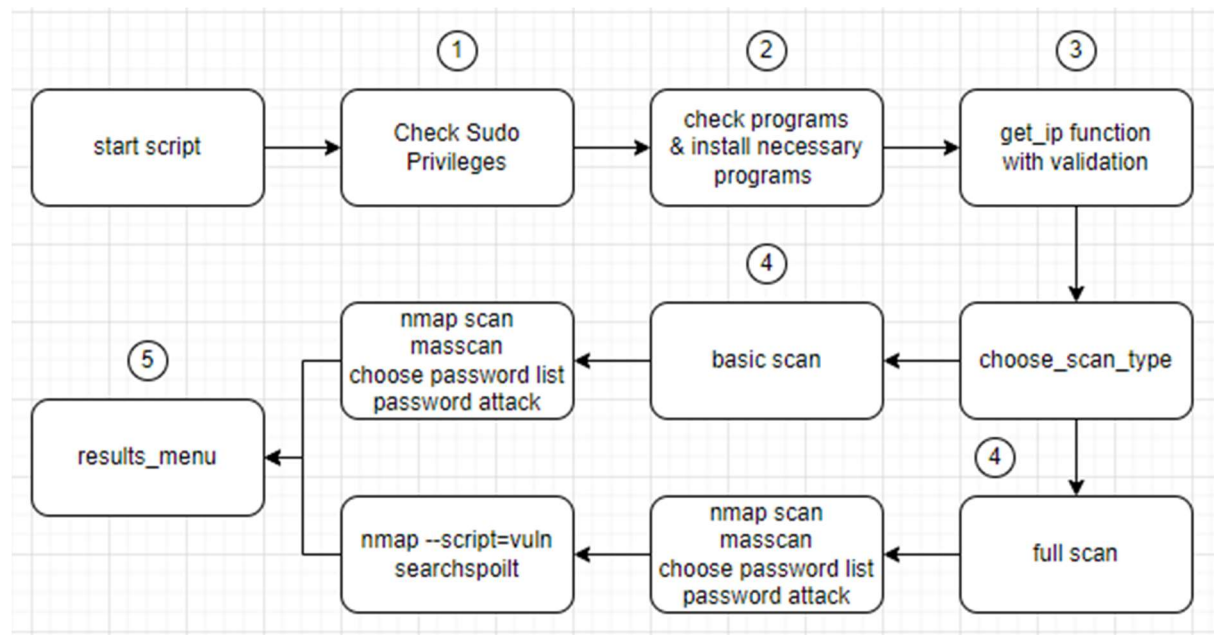
To present the importance of day-zero awareness and service updates in a more engaging manner, the project use automation as not just a response but as a proactive measure to strengthen defences before threats emerge as well as early vulnerability detection and password strength assessment to safeguard the network infrastructures.

The project's primary objective is to develop a versatile automation script that operates seamlessly on both Windows and Linux operating systems. This script will encompass five core functionalities:

1. **Getting the User Input:** It gets input for the network to scan and create a directory to store the information.
2. **Weak Credentials:** The script retrieves and displays results from hydra, passwords found on which services.
3. **Mapping Vulnerabilities:** if full was chosen, display potential vulnerabilities via NSE and Searchsploit.
4. **Log Results:** display each stage in the terminal, allow user to grep for information as well as a choice of whether to save the file or zipping.

This project script provides a comprehensive and user-friendly approach to network vulnerability assessment. It simplifies the process, allowing even novice users to conduct thorough scans and analyse results effectively. By following the documentation and using the script with the appropriate parameters, you can gain valuable insights into potential vulnerabilities in your network and take necessary measures to mitigate risks.

Diagram of script



#### Function of check sudo & check\_programs

Checks if script is running at sudo and installs a set of predefined cybersecurity tools.

Utilizes an associative array to manage the list of necessary tools, including Nmap, Hydra, Masscan, Medusa, and Searchsploit.

Iterates over the array, calling the install function for each tool, ensuring each is installed.

Then it calls for clone\_seclist function to clone the SecLists repository from github, to be used on the choose password list function (user & password list)

## 1. Getting the User Input

get\_ip function will prompts user for input. Then it will get processed at the validate function.

The input can be an IP address or a range in CIDR notation. It will be stored as network.

The second prompts user for input for the output directory, the location will store all the results from the scanning process, including logs, reports, and any extracted data.

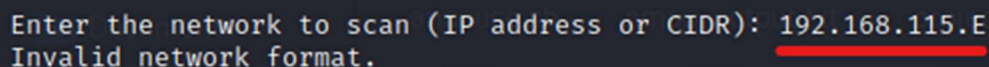
```
# Get network and output directory name
get_ip() {
    # Prompts user for network and output directory, creates directory
    read -p "Enter the network to scan (IP address or CIDR): " network #ask for input for to scan
    #network=192.168.115.133 #debug.
    if ! validate "$network"; then #call validate function.
        exit 1
    fi
    #Create dir
    read -p "Enter a name for the output directory: " output_dir #input will create the dir to store the outputs
    #mkdir -p "$output_dir"
}
```

Figure 1.1 A Screenshot of get\_ip()

The validate function will run input from \$network against regular expressions that match standard IP address formats. If the input does not according to the patterns, the script will echo "Invalid network format" and exit.

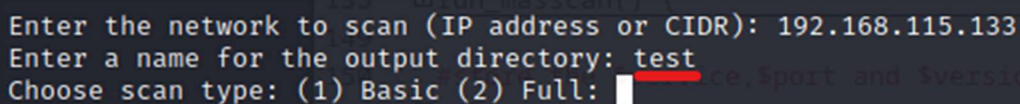
```
# Validate network input
validate() {
    if [[ $1 =~ ^([0-9]{1,3}\.){3}[0-9]{1,3}/[0-9]{1,2}$ ]]; then # validation "xxx.xxx.xxx.xxx/xx"
        return 0
    elif [[ $1 =~ ^([0-9]{1,3}\.){3}[0-9]{1,3}$ ]]; then # validation "xxx.xxx.xxx.xxx"
        return 0
    else
        echo "Invalid network format."
        return 1
    fi
}
```

Figure 1.1.1 A Screenshot of validate ()



A terminal window showing the prompt "Enter the network to scan (IP address or CIDR):" followed by the input "192.168.115.E". The output is "Invalid network format." The input "192.168.115.E" is underlined in red.

Figure 1.1.2 A Screenshot of validation with wrong IP/CIDR format



A terminal window showing the prompt "Enter the network to scan (IP address or CIDR):" followed by the input "192.168.115.133". The next prompt is "Enter a name for the output directory:" followed by the input "test". The next prompt is "Choose scan type: (1) Basic (2) Full:" followed by the input "1". The input "192.168.115.133" is underlined in red.

Figure 1.1.3: A Screenshot of get\_ip() passing ip validation and asking name for the output\_dir

After inputting the name for the output\_dir, the script will start choose\_scan\_type function, the user will be presented with a prompt to select the scan type:

The script waits for the user input and then proceeds based on the selection:

Input "1" initiates a Basic scan sequence.

Input "2" triggers the Full scan sequence.

Any other input results in an error message, and the function re-invokes itself for a correct input.

```
# Choose scan type
choose_scan_type() {
    # Prompts user to choose between Basic or Full scan
    read -p "Choose scan type: (1) Basic (2) Full: " scan_type #ask for basic or full 1 or 2
    echo ""
    if [[ $scan_type == "1" ]]; then
        echo "Running Nmap scan..."
        nmap_output+="Running Nmap scan..." #to be save in the log
        echo ""
        #nmap_output=$(nmap -sV $network | tee /dev/tty)
        nmap_output=$(nmap -sV -p- $network | tee /dev/tty) #to scan all ports
        echo ""

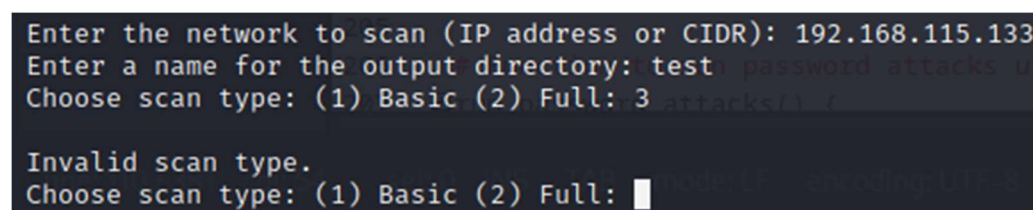
        run_masscan #run masscan
        selected_ports "$nmap_output" # stores the list of selected ports
        choose_password_list #this function choose password
        run_password_attacks #run hydra

    elif [[ $scan_type == "2" ]]; then
        echo "Running Nmap scan..."
        echo ""
        #nmap_output=$(nmap -sV $network | tee /dev/tty)
        nmap_output=$(nmap -sV -p- $network | tee /dev/tty) #to scan all ports
        echo ""

        run_masscan
        selected_ports "$nmap_output"
        choose_password_list
        run_password_attacks
        run_full_scan_analysis # run nmap --script=vuln -p
        searchsploit # run searchsploit

    else
        echo "Invalid scan type." # any other input will become this.
        choose_scan_type #dont let it break out.
    fi
}
```

Figure 1.2.1: A Screenshot of choose\_scan\_type ()



```
Enter the network to scan (IP address or CIDR): 192.168.115.133
Enter a name for the output directory: test
Choose scan type: (1) Basic (2) Full: 3
Invalid scan type.
Choose scan type: (1) Basic (2) Full: 1
```

Figure 1.2.2: if receive input other than 1 or 2 will echo Invalid scan type. And ask for input again.

After selecting scan type, the script will start the Nmap scan and appends this information to the nmap\_output log variable. It then executes the chosen type of scan, with each scan type branching into specific functions:

For the Basic scan, after running "nmap -sV \$network" it will call the function "run\_masscan", "selected\_ports", "choose\_password\_list", and "run\_password\_attacks".

For the Full scan, the same functions will be called, but with the addition of “run\_full\_scan\_analysis” function and “searchsploit” function for the comprehensive vulnerability assessment.

```
Running Nmap scan ...
Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-28 16:09 EST
Nmap scan report for msf (192.168.115.133)
Host is up (0.0018s latency).
Not shown: 65506 closed tcp ports (reset)
PORT      STATE SERVICE        VERSION
21/tcp    open  ftp            vsftpd 2.3.4
22/tcp    open  ssh            OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
23/tcp    open  telnet         Linux telnetd
25/tcp    open  smtp           Postfix smtpd
80/tcp    open  http           Apache httpd 2.2.8 ((Ubuntu) DAV/2)
111/tcp   open  rpcbind        2 (RPC #100000)
139/tcp   open  netbios-ssn    Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn    Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
512/tcp   open  exec           netkit-rsh rexecd
513/tcp   open  login          netkit-rsh rlogind
514/tcp   open  tcpwrapped
1099/tcp  open  java-rmi       GNU Classpath grmiregistry
1524/tcp  open  bindshell      Metasploitable root shell
2049/tcp  open  nfs            2-4 (RPC #100003)
2121/tcp  open  ftp            ProFTPD 1.3.1
3306/tcp  open  mysql          MySQL 5.0.51a-3ubuntu5
3632/tcp  open  distccd        distccd v1 ((GNU) 4.2.4 (Ubuntu 4.2.4-1ubuntu4))
3632/tcp  open  postgresql     PostgreSQL DB 8.3.0 - 8.3.7
5900/tcp  open  vnc            VNC (protocol 3.3)
6000/tcp  open  X11            (access denied)
6667/tcp  open  irc            UnrealIRCd
6697/tcp  open  irc            UnrealIRCd
8009/tcp  open  ajp13          Apache Jserv (Protocol v1.3)
8180/tcp  open  http           Apache Tomcat/Coyote JSP engine 1.1
8787/tcp  open  drb            Ruby DRB RMI (Ruby 1.8; path /usr/lib/ruby/1.8/drbb)
40563/tcp open  nlockmgr       1-4 (RPC #100021)
43202/tcp open  java-rmi       GNU Classpath grmiregistry
48063/tcp open  status         1 (RPC #100024)
53868/tcp open  mountd         1-3 (RPC #100005)
MAC Address: 00:0C:29:32:B7:E3 (VMware)
Service Info: Hosts: metasploitable.localdomain, irc.Metasploitable.LAN; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 132.77 seconds
```

Figure 1.2.3: A Screenshot of nmap result

The run\_masscan function in the script initiates a scan using Masscan to identify open UDP ports across the entire port range from 1 to 65535. The function echoes progress messages to the terminal and captures the output of Masscan, which is then redirected to both the standard output and a variable for logging purposes. After the scan is complete, a confirmation message is displayed. This allows for a rapid assessment of UDP services available on the target network, which is a crucial step in network security reconnaissance.

```
#run masscan
run_masscan() {
    # Group commands and direct all output to tee
    masscan_output=$(
        {
            echo ""
            echo "Starting masscan to scan udp ports..."
            echo ""
            #masscan -pU:1-1000 --rate 1000 "$network"
            masscan -pU:1-65535 --rate 1000 "$network" #scan all ports
            echo ""
            echo "Masscan scan completed."
        } 2>&1 | tee /dev/tty # Redirect both stdout and stderr
    )
}
```

Figure 1.3.1: A Screenshot of run\_masscan function

```
Starting masscan to scan udp ports ...rd attacks() {
Starting masscan 1.3.2 (http://bit.ly/14GZzcT) at 2023-12-28 21:11:14 GMT
Initiating SYN Stealth Scan
Scanning 1 hosts [65535 ports/host] scan analysis() {
Discovered open port 137/udp on 192.168.115.133
Masscan scan completed.
```

Figure 1.3.2: A Screenshot of run\_masscan result

## Selected Ports Functionality

The purpose of the `selected_ports` function is to parse the output from the Nmap scan and extract valuable data regarding open services, the ports, and versions.

The function reads the `nmap_output` through each line of the Nmap output, searching for patterns that match open TCP services. If it matches the Bash's regex it will capture the service name, port number, and service version. This function prepares this data for subsequent steps, such as vulnerability analysis and password attacks.

In addition to data extraction, the function has a filtering mechanism based on the predefined `selected_services` array, which includes services like SSH, RDP, FTP, and Telnet.

The captured information for these selected services is then stored in the `selected_ports_info` associative array for use in later stages of the penetration testing process.

```
#store the $service,$port and $version
selected_ports() {
    local nmap_output="$1"

    while read -r line; do
        if [[ $line =~ ^([0-9]+)/tcp[[:space:]]+open[[:space:]]+([a-zA-Z0-9_.-]+)[[:space:]]+(.+)$ ]]; then
            local port=${BASH_REMATCH[1]}
            local service=${BASH_REMATCH[2]}
            local version=${BASH_REMATCH[3]}

            # Check if the service is in the selected services list
            if [[ " ${selected_services[*]} " =~ " $service " ]]; then
                selected_ports_info["$service:$port"]="$version"
                #echo "Service: $service, Port: $port, Version: $version"
                #check if ${selected_services[*]} manage to catch the $service,$port and $version
            fi
        fi
    done <<< "$nmap_output"
}
```

Figure 1.4.1: A Screenshot of Selected Ports Function

```
echo "Service: $service, Port: $port, Version: $version"
```

```
Service: ftp, Port: 21, Version: vsftpd 2.3.4
Service: ssh, Port: 22, Version: OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
Service: telnet, Port: 23, Version: Linux telnetd
Service: ftp, Port: 2121, Version: ProFTPD 1.3.1
```

Figure 1.4.2: A Screenshot \$service, \$port and \$version working and printed the data correctly.

## choose\_password\_list functionality

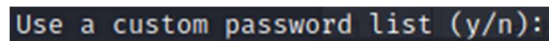
The purpose of “`choose_password_list`” function is to provide the penetration tester to specify a custom password list for brute force attacks or to proceed with a default list. When the function is triggered, the function will prompt the user to decide whether to use a custom password list. If the user chooses custom list, then the script will request the path to the list.

```
echo -e "\nuser\nmsfadmin\nservice\nnroot\npostgres\nftp\nanonymous\npassword" > msfadmin.txt
echo " Password File created at $(pwd)/msfadmin.txt"
default_location=$(pwd)/msfadmin.txt
```



This command creates a file named msfadmin.txt in the current working directory. The file contains a list of usernames, starting with a blank line. The -e flag in the echo command enables interpretation of backslash escapes like \n for new lines.

This command will assign the default\_location as \$(pwd)/msfadmin.txt(used as default password list, the input will be validated to ensure the specified file exists and is accessible. If the file is not found, the script defaults to a standard password list set.



```
Use a custom password list (y/n):
```

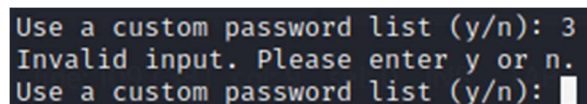
Figure 1.5.1: A Screenshot of choosing to use custom password list

The script requires a 'y' for 'yes' or 'n' for 'no' response. Any other input triggers an error message, and the prompt is displayed again until valid input is received. If the user inputs 'y', the function requests the path to the custom password list and checks for its existence. If the file is not found or 'n' is entered, the script informs the user that it will revert to the default password list and provides its location.



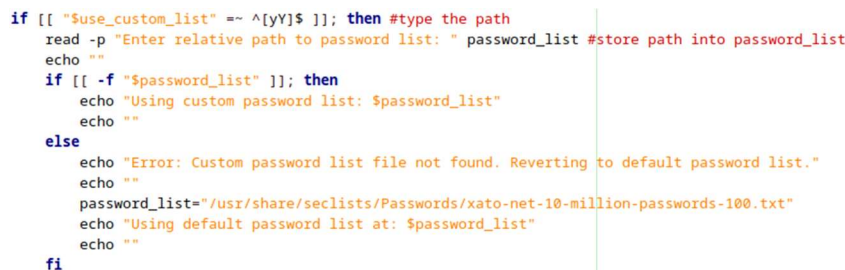
```
while true; do
  read -p "Use a custom password list (y/n): " use_custom_list #ask for y/ or *
  case $use_custom_list in
    [yY][nN]) break ;;
    *) echo "Invalid input. Please enter y or n." ;;
  esac
done
```

Figure 1.5.2: User Prompt and Validation in choose\_password\_list Function.



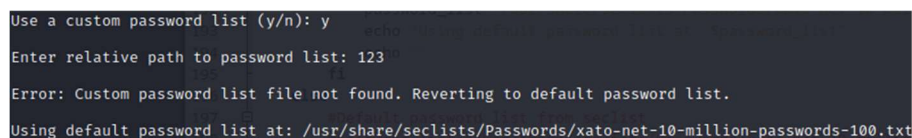
```
Use a custom password list (y/n): 3
Invalid input. Please enter y or n.
Use a custom password list (y/n):
```

Figure 1.5.3: A Screenshot of wrong input and echo “Invalid input”



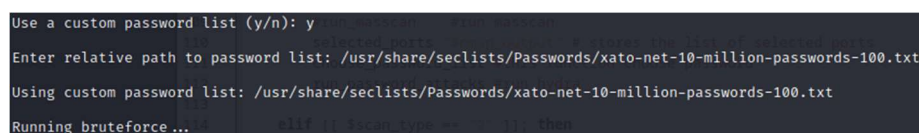
```
if [[ "$use_custom_list" =~ ^[yY]$ ]]; then #type the path
  read -p "Enter relative path to password list: " password_list #store path into password_list
  echo ""
  if [[ -f "$password_list" ]]; then
    echo "Using custom password list: $password_list"
    echo ""
  else
    echo "Error: Custom password list file not found. Reverting to default password list."
    echo ""
    password_list="/usr/share/seclists/Passwords/xato-net-10-million-passwords-100.txt"
    echo "Using default password list at: $password_list"
    echo ""
  fi
fi
```

Figure 1.5.4: Code Logic in choose\_password\_list Function for Custom Password File Verification



```
Use a custom password list (y/n): y
Enter relative path to password list: 123
Error: Custom password list file not found. Reverting to default password list.
Using default password list at: /usr/share/seclists/Passwords/xato-net-10-million-passwords-100.txt
```

Figure 1.5.5: A Screenshot of inputting 123 to test the script. File not found so the default list will be chosen.



```
Use a custom password list (y/n): y
Enter relative path to password list: /usr/share/seclists/Passwords/xato-net-10-million-passwords-100.txt
Using custom password list: /usr/share/seclists/Passwords/xato-net-10-million-passwords-100.txt
Running bruteforce ...
```

Figure 1.5.6: A Screenshot of code running and proceed to the next function “run\_password\_attacks”



## 2. Weak Credentials

The `run_password_attacks` function will utilize the Hydra to perform password attacks against the services and ports that were previously discovered and recorded by the `selected_ports` function.

The `service_count` maintains a count of how many services have been processed. If no services are found (i.e., `service_count` is zero), it outputs a message stating that there are no known services running for brute-force attacks. Otherwise, it appends the results of each Hydra attack to the `hydra_output` variable, which can later be reviewed or saved.

```
# Function to run password attacks using Hydra
run_password_attacks() {
    local service_count=0
    echo "Running bruteforce..."
    hydra_output+="Running bruteforce...\n"
    echo ""

    # Loop through the selected ports info
    for key in "${!selected_ports_info[@]}; do
        IFS=: read -r service port <<< "$key"
        local version=${selected_ports_info[$key]} #get version

        local message="Attacking '$service' (Version: $version) on port '$port'"\n"
        echo -e "$message" #if never add -e the \n will be printed too.
        hydra_output+=" $message\n"

        loop_hydra_output=$(hydra -L $userlist -P $password_list -s $port -t 16 $service://$network 2>&1 | tee /dev/tty)
        hydra_output+=" $loop_hydra_output\n" # Append each output to the variable
        ((service_count++))
        echo ""
    done

    if [[ $service_count -eq 0 ]]; then #if equal to 0 then it will echo nothing
        echo "No known services running for bruteforce."
        hydra_output+="No known services running for bruteforce."
        echo ""
    fi
}
```

Figure 2.1: A Screenshot of the `run_password_attacks` function.

Hydra attempts to log in to the service by iterating over the username and password lists, attempting to find valid credentials.

```
Running bruteforce ...
Attacking 'ftp' (Version: vsftpd 2.3.4) on port '21'
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-bin
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-12-29 10:46:39
[WARNING] Restorefile (you have 10 seconds to abort... (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 64 login tries (l:8/p:8), ~4 tries per task
[DATA] attacking ftp://192.168.115.133:21/
[21][ftp] host: 192.168.115.133 login: msfadmin password: msfadmin
[21][ftp] host: 192.168.115.133 login: user password: user
[21][ftp] host: 192.168.115.133 login: service password: service
[21][ftp] host: 192.168.115.133 login: postgres password: postgres
[21][ftp] host: 192.168.115.133 login: ftp password: msfadmin
[21][ftp] host: 192.168.115.133 login: ftp password: root
[21][ftp] host: 192.168.115.133 login: ftp password: user
[21][ftp] host: 192.168.115.133 login: ftp password: postgres
[21][ftp] host: 192.168.115.133 login: ftp password: service
[21][ftp] host: 192.168.115.133 login: ftp password: ftp
[21][ftp] host: 192.168.115.133 login: ftp password: anonymous
[21][ftp] host: 192.168.115.133 login: ftp password: password
[21][ftp] host: 192.168.115.133 login: anonymous password: msfadmin
[21][ftp] host: 192.168.115.133 login: anonymous password: user
[21][ftp] host: 192.168.115.133 login: anonymous password: service
[21][ftp] host: 192.168.115.133 login: anonymous password: root
[21][ftp] host: 192.168.115.133 login: anonymous password: postgres
[21][ftp] host: 192.168.115.133 login: anonymous password: ftp
[21][ftp] host: 192.168.115.133 login: anonymous password: anonymous
[21][ftp] host: 192.168.115.133 login: anonymous password: password
1 of 1 target successfully completed, 20 valid passwords found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2023-12-29 10:47:00
```

Figure 2.2: A Screenshot of hydra working as intended.

```
Running bruteforce ...
No known services running for bruteforce.
```

Figure 2.3: A Screenshot of hydra echo “No known services running for bruteforce” when there is no services to bruteforce

For each service, the function executes Hydra with the following parameters:

-L: This flag specifies the list of usernames Hydra will use to attempt logins. It's crucial for the script to automate the login process across multiple user accounts.

-P: Similar to the -L flag, but for passwords. This list is pivotal in testing the strength of passwords across services.

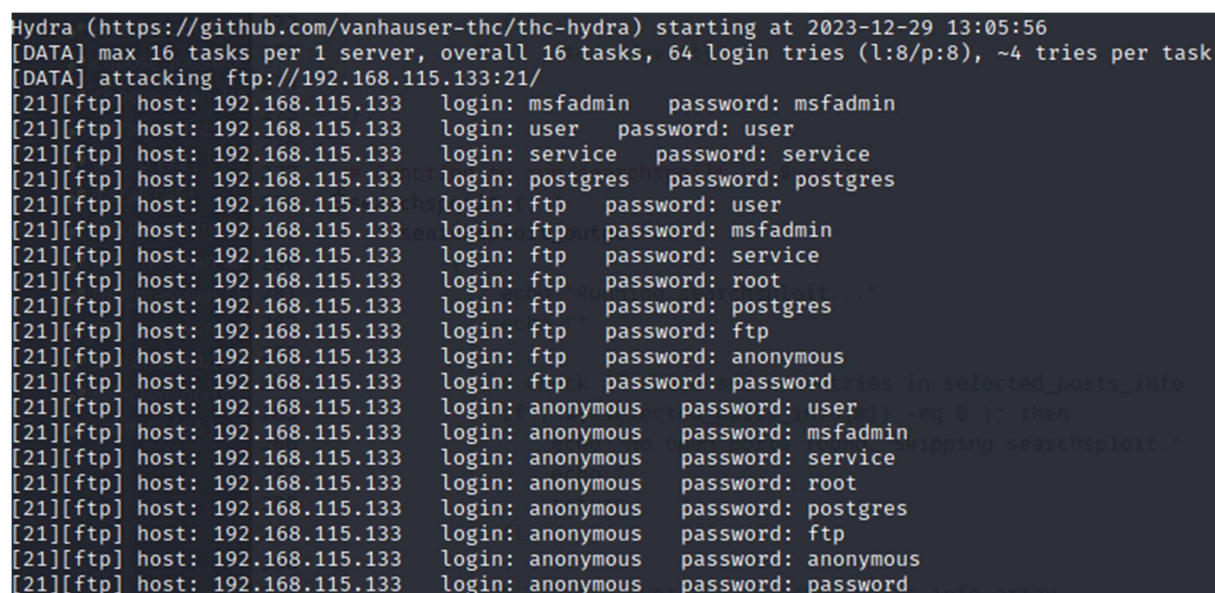
-s: Designates the port number for the service Hydra is targeting. Since different services operate on different ports, this flag ensures that Hydra attempts to log in to the correct service.

-t: Sets the number of concurrent connections Hydra can make, which impacts the speed and efficiency in the event of rate-limiting by the target services, which is a defense mechanism where the service limits the number of login attempts over a period, Hydra might experience timeouts or other errors.

The script does not automatically handle these cases; manual intervention is necessary to adjust the rate of the attack (-t flag for threads) or to implement a delay between attempts of the brute force attack.

```
loop_hydra_output=$(hydra -L $userlist -P $password_list -s $port -t 16 $service://$network 2>&1 | tee /dev/tty)
hydra_output+="$loop_hydra_output\n" # Append each output to the variable
((service_count++))
```

Figure 2.4: A Screenshot the hydra and the flags



```
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-12-29 13:05:56
[DATA] max 16 tasks per 1 server, overall 16 tasks, 64 login tries (l:8/p:8), ~4 tries per task
[DATA] attacking ftp://192.168.115.133:21/
[21][ftp] host: 192.168.115.133 login: msfadmin password: msfadmin
[21][ftp] host: 192.168.115.133 login: user password: user
[21][ftp] host: 192.168.115.133 login: service password: service
[21][ftp] host: 192.168.115.133 login: postgres password: postgres
[21][ftp] host: 192.168.115.133 login: ftp password: user
[21][ftp] host: 192.168.115.133 login: ftp password: msfadmin
[21][ftp] host: 192.168.115.133 login: ftp password: service
[21][ftp] host: 192.168.115.133 login: ftp password: root
[21][ftp] host: 192.168.115.133 login: ftp password: postgres
[21][ftp] host: 192.168.115.133 login: ftp password: ftp
[21][ftp] host: 192.168.115.133 login: ftp password: anonymous
[21][ftp] host: 192.168.115.133 login: ftp password: password
[21][ftp] host: 192.168.115.133 login: anonymous password: user
[21][ftp] host: 192.168.115.133 login: anonymous password: msfadmin
[21][ftp] host: 192.168.115.133 login: anonymous password: service
[21][ftp] host: 192.168.115.133 login: anonymous password: root
[21][ftp] host: 192.168.115.133 login: anonymous password: postgres
[21][ftp] host: 192.168.115.133 login: anonymous password: ftp
[21][ftp] host: 192.168.115.133 login: anonymous password: anonymous
[21][ftp] host: 192.168.115.133 login: anonymous password: password
```

The script takes a conservative approach to password attacks to minimize the risk of account lockouts or triggering intrusion detection systems. However, if false positives occur (where the script reports a successful login that is not actually valid), these should be manually verified through additional testing.

The size of the username and password lists used in brute-force or dictionary attacks with tools like Hydra has a direct impact on the attack's effectiveness, duration, and the network's response.

### 3. Mapping Vulnerabilities

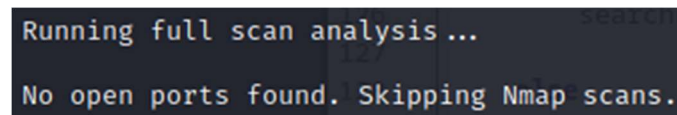
The `full_scan_analysis` function will apply the results from “`selected_ports_info`” and applies the ports & services found to Nmap Scripting Engine (NSE) to identified vulnerabilities.

If no services are available (e.g., no open ports were found), the function terminates early, indicating that no scans will be performed.

```
run_full_scan_analysis() {
    full_scan_output=$(
        {
            echo "Running full scan analysis..."
            echo ""

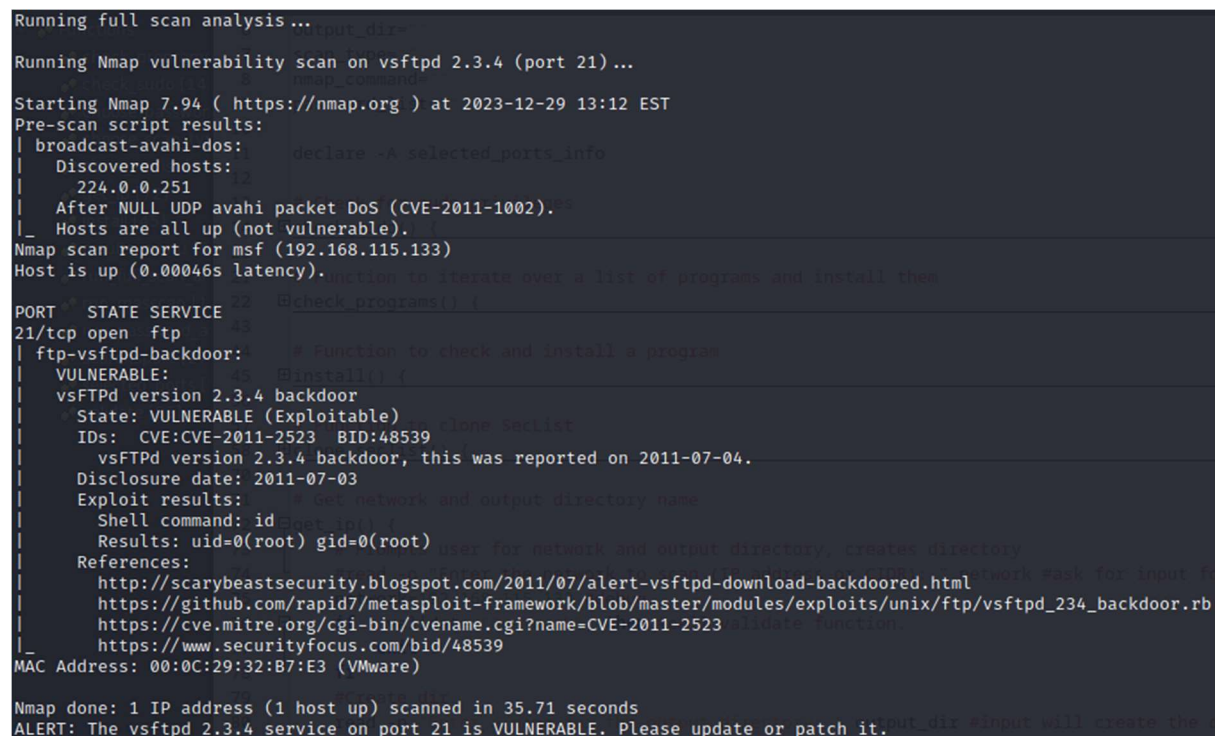
            #if no port there is no spoon
            if [ ${#selected_ports_info[@]} -eq 0 ]; then
                echo "No open ports found. Skipping Nmap scans."
                echo ""
                return
            fi
        }
    )
}
```

Figure 3.1 : Code Logic in `full_scan_analysis` Function if no ports were found. (`selected_ports = 0`)



```
Running full scan analysis...
No open ports found. Skipping Nmap scans.
```

Figure 3.2 : A Screenshot of code running when no open ports were found.



```
Running full scan analysis...
Running Nmap vulnerability scan on vsftpd 2.3.4 (port 21)...

Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-29 13:12 EST
Pre-scan script results:
| broadcast-avahi-dos: declare -A selected_ports_info
| Discovered hosts:
| 224.0.0.251
| After NULL UDP avahi packet DoS (CVE-2011-1002).
|_ Hosts are all up (not vulnerable).
Nmap scan report for msf (192.168.115.133)
Host is up (0.00046s latency).

PORT      STATE SERVICE
21/tcp    open  ftp
| ftp-vsftpd-backdoor:
| VULNERABLE:
| vsFTPD version 2.3.4 backdoor
| State: VULNERABLE (Exploitable)
| IDs: CVE:CVE-2011-2523 BID:48539
| vsFTPD version 2.3.4 backdoor, this was reported on 2011-07-04.
| Disclosure date: 2011-07-03
| Exploit results:
| Shell command: id
| Results: uid=0(root) gid=0(root)
| References:
| http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html
| https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/unix/ftp/vsftpd_234_backdoor.rb
| https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2523
| https://www.securityfocus.com/bid/48539
|_
MAC Address: 00:0C:29:32:B7:E3 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 35.71 seconds
ALERT: The vsftpd 2.3.4 service on port 21 is VULNERABLE. Please update or patch it.
```

Figure 3.3 : A Screenshot of the NSE result with vulnerability were found.

When the keyword 'VULNERABLE' is detected in the scan output, the function constructs an alert message indicating that the service and its corresponding port have known vulnerabilities. This message also prompts for an update or patch to mitigate the risk.

```
#check if the word VULNERABLE is present
if echo "$service_output" | grep -q 'VULNERABLE'; then #if got VULNERABLE then
    vulnerability_summary+="ALERT: The $version service on port $port is VULNERABLE. Please update or patch it.\n"
fi
```

```
ALERT: The vsftpd 2.3.4 service on port 21 is VULNERABLE. Please update or patch it.
```

Figure 3.4 : A Screenshot of code running when the keyword 'VULNERABLE' is detected. It will echo the \$version and the \$port

```
#list of vulnerability
if [ -z "$vulnerability_summary" ]; then
    echo "There is no vulnerability found."
    echo ""
else
    # Print the list of vulnerability
    echo -e "$vulnerability_summary"
    echo ""
fi
```

The script checks if the vulnerability\_summary variable is empty, which would indicate that no vulnerabilities were found during the scan. If the variable is empty, the script informs the user that there are no vulnerabilities with the message "There is no vulnerability found."

```
Running full scan analysis ...
Running Nmap vulnerability scan on OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0) (port 22) ...
Starting Nmap 7.94 ( https://nmap.org ) at 2023-12-29 13:25 EST
Pre-scan script results:
| broadcast-avahi-dos:
|   Discovered hosts:
|     224.0.0.251
|   After NULL UDP avahi packet DoS (CVE-2011-1002).
|_ Hosts are all up (not vulnerable).
Nmap scan report for msf (192.168.115.133)
Host is up (0.00031s latency).

PORT      STATE SERVICE
22/tcp    open  ssh
MAC Address: 00:0C:29:32:B7:E3 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 34.41 seconds
There is no vulnerability found.
```

Figure 3.5 : A Screenshot of a successful NSE scan but with vulnerability found.



## searchsploit

The searchsploit function also will retrieve results from “selected\_ports\_info “ use the version data, the searchsploit command is executed with the -e flag, which stands for exact match, to find exploits that closely match the version of the service found. The output of searchsploit is then echoed to the terminal and captured within the searchsploit\_output variable. If no services are available (e.g., no open ports were found), the function terminates early, indicating that no scans will be performed.

```
Running searchsploit ...  
No open ports found. Skipping searchsploit.
```

Figure 3.6 : A Screenshot of code running when no open ports were found.

```
Running searchsploit ...  
vsftpd 2.3.4  
Running searchsploit to check vulnerability for ftp (version vsftpd 2.3.4) on port 21 ...  
Exploit Title  
vsftpd 2.3.4 - Backdoor Command Execution  
vsftpd 2.3.4 - Backdoor Command Execution (Metasploit)  
Shellcodes: No Results  
Papers: No Results
```

Figure 3.7 : A Screenshot of the searchsploit with vulnerability were found.

```
Running searchsploit ...  
OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)  
Running searchsploit to check vulnerability for ssh (version OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)) on port 22 ...  
Exploits: No Results  
Shellcodes: No Results  
Papers: No Results
```

Figure 3.8 : A Screenshot of the searchsploit but no vulnerability were found.

#### 4. Log Results

The results\_menu function serves as the interactive end-point of the script. It compiles all gathered data into the combined\_output variable, offering the user a concise interface for post-scan options.

```
combined_output="${check_programs_output}\n\n${nmap_output}
```

The variable combined\_output concatenates the outputs from various stages, such as program checks and network scans, ensuring all relevant data is accessible in one place.

```
Choose an option:
1. Search results
2. Save results
3. Exit
```

The looped menu presents three options, allowing users to search within the combined results, save them to a file, or exit the script.

```
while true; do
    echo "Choose an option:"
    echo "1. Search results"
    echo "2. Save results"
    echo "3. Exit"
    echo ""
    read -p "Enter your choice: " option
```

Figure 4.1 : shows the initial user interface of the results\_menu function. It depicts the options presented to the user for interacting with the scan results.

```
case "$option" in
    1)
        read -p "Enter search term: " search_term #ask for things to grep
        echo "$combined_output" | grep -i "$search_term" ;;
```

When the user selects option 1 from the results\_menu, they are prompted to enter a term they wish to search for within the scan results.

The grep -i command performs a case-insensitive search, ensuring that matches are found regardless of how the search term is capitalized.

The search results are then displayed to the user, showing only the portions of the combined output that contain the search term.

```
Enter your choice: 1
Enter search term: port
Nmap scan report for msf (192.168.115.133)
PORT      STATE SERVICE VERSION
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Running Nmap vulnerability scan on OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0) (port 22)...
Nmap scan report for msf (192.168.115.133)
PORT      STATE SERVICE
Running searchsploit to check vulnerability for ssh (version OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)) on port 22 ...
```

The search feature allows users to quickly locate specific information within the scan results. By entering a search term, the script filters the combined output and displays any matching lines.



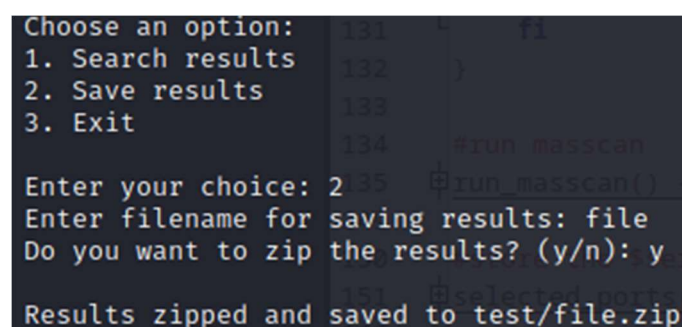
This portion of the script provides the user with the functionality to save the combined output of the scans to a file, where it prompts the user to enter a filename and decide whether to compress the results into a ZIP file. The script includes a loop to ensure valid input is received before proceeding with the saving process.

```
2)
read -p "Enter filename for saving results: " save_file #ask for filename
while true; do
    read -p "Do you want to zip the results? (y/n): " zip_choice
    echo ""
    case "$zip_choice" in
        [Yy]* )
            # Ensure the output directory exists
            mkdir -p "$output_dir"
            # Save and zip the results
            if printf "%b" "$combined_output" > "$output_dir/$save_file.txt" && \
                zip -j "$output_dir/$save_file.zip" "$output_dir/$save_file.txt" > /dev/null; then
                rm "$output_dir/$save_file.txt"
                echo "Results zipped and saved to $output_dir/$save_file.zip"
                echo ""
            else
                echo "Failed to zip the results. Check if the zip utility is installed." #if fail, unlikely
            fi
            break
        ;;
        [Nn]* )
            # Ensure the output directory exists & output to $save_file.txt
            mkdir -p "$output_dir"
            printf "%b" "$combined_output" > "$output_dir/$save_file.txt"
            echo "Results saved to $output_dir/$save_file.txt"
            echo ""
            break
        ;;
        * )
            echo "Invalid input. Please enter y or n."
            echo ""
            ;;
    esac
done
;;
```

Figure 4.1 portion of the script provides the user with the functionality to save the combined

If the user chooses to zip the results, the script ensures that the specified output directory exists using `mkdir -p`. It then saves the combined output to a `.txt` file and the results are compressed into a `.zip` file using the `zip` command.

The original `.txt` file is removed, leaving only the compressed `.zip` file & the user will be notified of the successful zipping and location of the saved `.zip` file.



```
Choose an option:
1. Search results
2. Save results
3. Exit

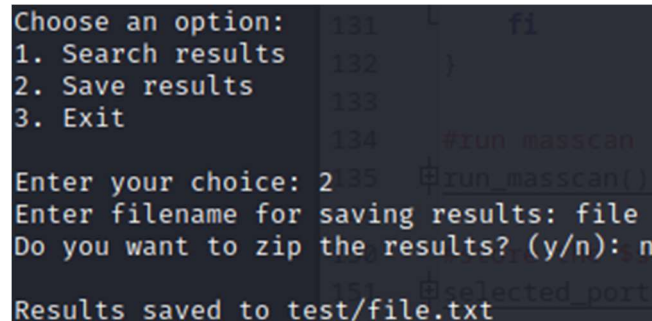
Enter your choice: 2
Enter filename for saving results: file
Do you want to zip the results? (y/n): y
Results zipped and saved to test/file.zip
```

Figure 4.2 Results zipped and saved to test/file.zip

If the user chooses not to zip the results, the script ensures that the specified output directory exists by using `mkdir -p`.

Then the combined output is saved to a .txt file in the specified directory.

The user is informed about the location of the saved .txt file.



```
Choose an option: 131      fi
1. Search results 132      }
2. Save results   133
3. Exit           134      #run masscan
Enter your choice: 2 135      @run masscan()
Enter filename for saving results: file
Do you want to zip the results? (y/n): n
Results saved to test/file.txt
```

Figure 4.3 Results saved to test/file.txt

The script will loop back to the main menu, prompting the user to either

1) search results

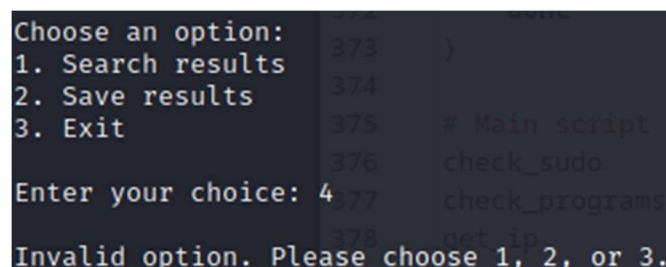
2) save results

3) Exit

```
3)
    echo "Exiting..."
    exit 0 ;;
*)
    echo ""
    echo "Invalid option. Please choose 1, 2, or 3."
    echo "";;
esac
done
```

If the user does not choose 1, 2, or 3, the script prompts them to make a valid selection, enhancing the user interface by preventing unexpected script terminations and guiding the user towards a correct response.

If the user enters an invalid input when asked to zip the results, the script prompts them again.



```
Choose an option: 373      }
1. Search results 374
2. Save results   375      # Main script
3. Exit           376      check_sudo
Enter your choice: 4 377      check_programs
Invalid option. Please choose 1, 2, or 3. 378      get_ip
```

Figure 4.4 A Screenshot of the menu entering wrong input to test the script. Which echo "Invalid option. Please choose 1, 2, or 3."

## CONCLUSIONS

This Bash scripting project was a practical exercise in automating network security tasks. It involved leveraging tools like Nmap, Hydra, and Searchsploit to conduct network scans and identify vulnerabilities and weak passwords. The script's flexibility in offering basic and full scans allowed for adaptable security assessments.

User input validation and data management were critical to the script's effectiveness, ensuring accurate scans and organized results. Handling varying scan outcomes, such as the absence of vulnerabilities, was an important feature that added robustness to the tool.

Overall, the project highlighted the efficiency gains from automation in cybersecurity practices and the importance of a methodical approach to security assessments. It has been a valuable step in my ongoing journey to master cybersecurity automation.