Penetration Testing Project

TMagen773637.S11.ZX301

Ron Nisinboym

August 23, 2025

Overview

Project Description:

This project automates the process of network scanning, vulnerability assessment, and password strength evaluation. It scans a specified network for active hosts, identifies services, and checks for weak passwords. The script assesses known vulnerabilities and evaluates password security risks. Users can choose between basic or full scan options, with all results compiled into a single detailed report.

Technologies Used

- **Bash Scripting**: Used for automating the execution of commands, managing file paths, and automating the report generation.
- **Nmap:** Scans the network to identify active hosts and the versions of services running on them for vulnerability assessment.
- **Masscan (UDP Ports Scanning)**: Uses Masscan to rapidly scan large networks for open UDP ports.
- **SearchSploit**: Searches the Exploit-DB repository for known exploits related to the identified services and their versions.
- **Nmap NSE (Vulners)**: Cross-references scan results with a vulnerability database to identify known vulnerabilities in the target hosts.
- **Nmap NSE (Weak Password Brute Force)**: Uses Nmap's scripting engine to attempt brute-force password cracking on identified services, checking for weak passwords.

Detailed Workflow

1. **Initial Setup** The script begins by verifying root user privileges, then prompts the user to input the network range to scan and a folder name where the results will be saved. It automatically downloads user and password lists from a GitHub repository.
2. **Scan Configuration:** The user is prompted to select the scan level (basic/full). The basic scan will identify hosts, services, and assess weak passwords, while the full scan will additionally map vulnerabilities using SearchSploit and the Nmap Vulners script.
3. **Brute-Force Method Selection:** The user is given the option to choose a brute-force method for weak password detection, including specifying which usernames and passwords to use during the attack.
4. **Execution Stage:** With the user's input, the script executes the scan, displaying real-time progress in the terminal. Each tool's output is saved to a separate text file corresponding to each host. Additionally, a comprehensive report file is generated containing a summary of all hosts discovered and the results from each tool.
5. **Search Functionality:** Upon completion of data gathering, the full report is displayed in the terminal. The user is then prompted to search for specific strings or patterns within the report for further analysis.
6. **Zipping Results**: Finally, the user is given the option to zip the results for easy storage and sharing.

Design Principles

1. **Simplicity Over Decoration:** By reducing the use of bold colors, fancy fonts, and overly complex layouts, the script avoids visual clutter, providing an **"easy on the eyes" experience**.
2. **Clear, Readable Output**:
    a. The font choice is kept **simple and legible**, ensuring that the user can quickly read and interpret the different output, even over long periods of use.
    b. Colors are used sparingly, with only essential information (e.g. important information, warnings, errors and successes) highlighted using subtle color changes. This reduces eye strain while still providing the necessary emphasis on important results.

Credits

1. **Erel Regev (lecturer) –** for kickstarting the bash script, providing the initial framework and guidance by supplying the foundational script structure and key concepts which was later modified and expanded to my own vision of this project.
2. **ChatGPT** – The use of AI was made exclusively as a helping tool. Used to fix some of the errors and bugs occurring while writing the script, consulting in regards of the script's logic when more difficult steps were used(e.g. Methods, Pearl-regex), asking for general information(e.g. "*Xargs*", Delimiters, Internal Field Separators), and things I struggled to remember(e.g. certain flags). The choice of ChatGPT over Google, forums or other sources was made mostly to save time and get straight forward explanations for each piece of advice to keep learning from it.

Detailed Walkthrough Of The Script

This section provides a breakdown of the script's functionality, including detailed explanations of key code snippets, along with screenshots of the code and its practical execution.

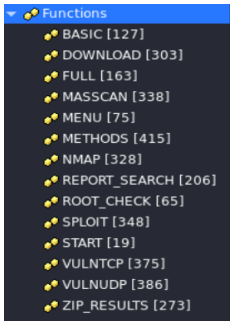
Global Variables -

In this screenshot we can see all the global variables that are being used by the script.
These variables **are not** nested in any function, to be available for any function that needs to call them.

```
############################
#COLORS AND FONTS:
GC='\e[32m' #GREEN
RC='\e[31m' #RED
CC='\033[0;36m' #CYAN
BOLD='\033[1m' #BOLD
NC='\e[0m' #DEFAULT

#OTHER VARIABLES:
LOCAL_IP=$(hostname -I | xargs) #defines local_ip variable without spaces
service="" #service + version
PORT="" #used to itterate between ports in a later defined array.
MAIN="$(pwd)" #shortcut for pwd
############################
```

Functions in general -

In this screenshot we can see a list of all functions that are being used by the script.
The idea is to encapsulate repetitive chunks of code or any code that has a unique behavior, to achieve ease of access and further modifying of specific functionality.
Therefore, all pieces of code in this script, **except** global variables, are nested inside functions.



Starting the script -

This function starts the script once the user tries to run the ".sh" file, immediately calling the "ROOT_CHECK" function to prevent any user who is not root from continuing to run it.

It then gathers users input for the initial setup of the script, using an "*If*" statement to avoid certain errors.

```
[[ "$network_range" != *.*.*.*/*?* ]]
```

This part of the *"if"* statement **doesn't** prevent all kinds of wrong input(e.g. letters), but rather checks that the pattern is correct, to minimize the chance for error.

The function calls the "ROOT_CHECK", "DOWNLOAD" and "BASIC" or "FULL" functions.

```
#Function that starts the script
function START()
{
    ROOT_CHECK #calling function to check root
    echo -e "What is the network range to scan?\n(e.g. 192.168.1.0/24) "
    read network_range
    if [ -z "$network_range" ] || [[ "$network_range" != *.*.*.*/*?* ]]; then #if variable returns empty or with wrong pattern
    echo -e "${RC}Invalid input. Use a format like 192.168.1.1/24${NC}\n"
    START  #restart script
    exit #used to avoid processing all previous instances of START in case of a wrong input
fi

    echo -e "\nPlease choose a name for the output directory: "
    echo -e "(***Choosing a name of an existing folder in $MAIN will delete the old one***)"
    read dir_name
    if [ -z "$dir_name" ]; then #if variable returns empty(not set)
        echo -e "${RC}No input was made. Try again.${NC}\n"
    START #start again if wrong input
    elif [ -d "$dir_name" ]; then #if such directory exists
        rm -rf "$dir_name" #remove
    fi

    mkdir "$dir_name"  #creates defined by user directory
    report_file="$dir_name/full_report.txt" #creates the full_report text file

    DOWNLOAD

    while true; do #letting user to choose scanning level
        echo ""
        read -p "Please choose a Basic or Full scanning level [B/F]: " LEVEL
        case $LEVEL in
            [bB]*)
                BASIC
                break  #exit the loop
                ;;
            [fF]*)
                FULL
                break  #exit the loop
                ;;
             *)
                echo -e "${RC}Invalid option, please try again.${NC}"
                ;;
        esac
    done
}
```

Checking for root user -

This function uses an *"if"* statement to check if the current user is a root user.

It is done to prevent a user who is not root to continue running the script, and to have elevated permissions for certain actions taking place(e.g. creation of folders and files).

```
#function that checks if user is root
function ROOT_CHECK(){
    USER=$(whoami)
    if [ "$USER" != "root" ] #if NOT root.
    then
        echo "You are not root. Exiting.."
        exit
    fi
}
```

Downloading lists -

This function runs "*if*" statements to check if the username and password lists exist inside the relevant folder.
If not, it initiates "*wget*" to download "raw" files from a Github repository.

```
#Function to download username and password lists for the script.
function DOWNLOAD () {
    echo -e "\nChecking for username/password lists in $MAIN..."

    #checks if the password list is missing
    if [ ! -f $MAIN/10k-most-common-passwords.txt ]; then
        echo "Downloading 10k-most-common-passwords.txt..."
        #download file from github, hide output.
        wget https://github.com/CookieBotXL/Weak_Passwords/raw/refs/heads/main/10k-most-common-passwords.txt -P $MAIN/ > /dev/null 2>&1
        echo -e "${GC}10k-most-common-passwords.txt has been downloaded successfully${NC}"
    else
        echo "10k-most-common-passwords.txt exists. Skipping download."
    fi

    #checks if the username file is missing
    if [ ! -f $MAIN/top-usernames-shortlist.txt ]; then
        echo "Downloading top-usernames-shortlist.txt..."
        #download file from github, hide output.
        wget https://github.com/CookieBotXL/Weak_Passwords/raw/refs/heads/main/top-usernames-shortlist.txt -P $MAIN/ > /dev/null 2>&1
        echo -e "${GC}top-usernames-shortlist.txt has been downloaded successfully${NC}"
    else
        echo -e "top-usernames-shortlist.txt exists. Skipping download."
    fi
}
```

Basic Scanning -

This function uses the command -

```
grep -Ev "(\.2$|\.254$|^$LOCAL_IP$)"
```

It is done to prevent the scanning of IP's that VMware sets for the default-gateway and DHCP by default, along with the local IP address of the user. **Consider editing this line if the script runs outside of a default VMware environment.**
It continues to a "*for*" loop, where it iterates over each discovered host, to run a set of different functions against it. When done, it shows the gathered data from the report file in the terminal, before calling the last two functions that lets the user choose to search inside the results and archive them.

This function calls the "MENU", "NMAP", "MASSCAN", "METHODS", "REPORT_SEARCH" and "ZIP_RESULTS" functions.

```
# Function to check reachable hosts and scan for services + weak password
function BASIC() {

    MENU

    #checking for reachable hosts | awking for IP only | excluding local-host and VM default IPs(gateway\dhcp)
    nmap $network_range -sn | awk '/Nmap scan report for/ {print $NF}' | grep -Ev "(\.2$|\.254$|^$LOCAL_IP$)" >> "$dir_name/ips.txt"

    if [ ! -s "$dir_name/ips.txt" ]; then #if ips.txt returns empty
        echo -e "${RC}No reachable hosts has been found. Exiting script.${NC}"
        exit 1
    fi

    for ip in $(cat $dir_name/ips.txt) #for each ip found do the following.
    do
        mkdir -p $dir_name/$ip #creating file containing list of found ips.
        echo -e "\n${GC}$ip was found${NC}\n"
        echo "==== $ip ====" >> "$report_file" #creating a header for the report file.

        NMAP #NMAP-realated processes

        MASSCAN #MASSCAN-realated processes

        METHODS #NSE_BRUTE-related processes

    done

    echo -e "Showing full report: \n"
    sleep 5 #giving the user time to read the above message
    cat -n $report_file #show report output
    echo -e "\n${CC}This output was saved to $report_file${NC}"

    REPORT_SEARCH
    ZIP_RESULTS
}
```

Full Scanning -

This function executes the same logic as the “BASIC” function with the addition of calling another three functions - “SPLOIT”, “VULNTCP”, and “VULNUDP”.

```
#Function that runs full scan including mapping vulnerablities
function FULL () {

    MENU

    #checking for reachable hosts | awking for IP only | excluding local-host and VM default IPs(gateway\dhcp)
    nmap $network_range -sn | awk '/Nmap scan report for/ {print $NF}' | grep -Ev "(\.2$|\.254$|^$LOCAL_IP$)" >> "$dir_name/ips.txt"

    if [ ! -s "$dir_name/ips.txt" ]; then #if ips.txt returns empty
        echo -e "${RC}No reachable hosts has been found. Exiting script.${NC}"
        exit 1
    fi

    for ip in $(cat $dir_name/ips.txt) #for each ip found do the following
    do

        mkdir -p $dir_name/$ip #creating file containing list of found ips
        echo -e "\n${GC}$ip was found${NC}\n"
        echo "==== $ip ====" >> "$report_file" #creating a header for the report file

        NMAP #NMAP-realated processes

        MASSCAN #MASSCAN-realated processes

        VULNUDP #NSE_VULN_UDP related processes

        VULNTCP #NSE_VULN_TCP-related processes

        SPLOIT #SEARCHSPLOIT-related processes

        METHODS #NSE_BRUTE-related processes
    done

    echo -e "Showing full report: \n"
    sleep 5 #giving user time to read the last message
    cat -n $report_file
    echo -e "\n${CC}This output was saved to $report_file${NC}"
    REPORT_SEARCH
    ZIP_RESULTS


}
```

Menu(Brute-Force) -

This function opens a menu where the user may choose the brute-force approach that will be executed later in the “METHODS” function, and provide the necessary details for it to run properly.

```
# Function that lets user define the Brute-Force method that will be applied to all services
function MENU() {
    METHOD=""

    while true; do
    echo -e "Please choose a brute-force method that will be used for all services found [CHOOSE A NUMBER]\n\
    [+] 1 - custom username + custom password list\n\
    [+] 2 - custom username + built-in password list\n\
    [+] 3 - custom username list + custom password list\n\
    [+] 4 - custom username list + built-in password list\n\
    [+] 5 - built-in users&password lists\n\
    [+] 0 - Exit"

    read -p "Enter your choice: " OPTIONS

    case $OPTIONS in #setting up method choices for each of the menu options to further execute via METHODS function
        1)  #custom username + custom password list
            read -p "Enter username: " USERNAME
            echo "Path to password list: "
            read -e PASS_PATH #read + autocomplete

            #checks if the password list file exists
            if [ ! -f "$PASS_PATH" ]; then
                echo -e "${RC}Error: Password list file not found at '$PASS_PATH', try again.${NC}\n"
                continue #restarts the loop
            fi
            METHOD="1"
            break #exit loop
            ;;
        2)  #custom username + built-in password list
            read -p "Enter username: " USERNAME
            METHOD="2"
            break #exit loop
            ;;
        3)  #custom username list + custom password list
            echo "Path to users list: "
            read -e USER_PATH #read + autocomplete

            #checks if the user list file exists
            if [ ! -f "$USER_PATH" ]; then
                echo -e "${RC}Error: User list file not found at '$USER_PATH', try again.${NC}\n"
                continue #restars the loop
            fi

            echo "Path to password list: "
            read -e PASS_PATH #read + autocomplete

            #checks if the password list file exists
            if [ ! -f "$PASS_PATH" ]; then
                echo -e "${RC}Error: Password list file not found at '$PASS_PATH', try again.${NC}\n"
                continue #restarts the loop
            fi
            METHOD="3"
            break #exit loop
            ;;

        4)  #custom username list + built-in password list
            echo "Path to users list: "
            read -e USER_PATH #read + autocomplete

            #checks if the user list file exists
            if [ ! -f "$USER_PATH" ]; then
                echo -e "${RC}Error: User list file not found at '$USER_PATH', try again.${NC}\n"
                continue #restarts the loop
            fi
            METHOD="4"
            break #exit loop
            ;;
        5)  #built-in users&password lists
            METHOD="5"
            break #exit loop
            ;;
        0)
            exit 1
            ;;
        *)
            echo -e "${RC}Invalid option. Please try again.\n${NC}"
            MENU
            return #used to avoid processing all previous instances of MENU in case of a wrong input
            ;;
    esac

    echo -e "Brute-force method chosen: $METHOD"
    echo -e "Scanning the network, please wait...\n"
    done
}
```

Methods(Brute-Force) -

This function uses the defined credentials and the corresponding method that was chosen in the "MENU" function, to execute an Nmap *brute.nse* attack against the relevant services.
It uses an array of port numbers tied to services names, that are passed to the chosen method along with the credentials choice of the user, to iterate over each service and find weak-passwords.
Finally, it greps and saves successful attempts to a separate text file for each service scanned, while also adding it to the report file.

```
#Function to use different brute-force methods, against every found service
function METHODS () {
        LOGIN_SERVICES="ssh ftp telnet rdp" #string of services to itterate through
        declare -A SERVICE_PORTS=( ["ftp"]=21 ["ssh"]=22 ["telnet"]=23 ["rdp"]=3389 ) #array of ports for each service.

        echo -e "Checking for weak passwords against common services. Please wait...\n"
        echo -e "==== BRUTE NSE ====" >> "$report_file" #header
        for service in $LOGIN_SERVICES #loop through each service and check if it is found in the nmap results
        do
            if grep -q "$service" $dir_name/$ip/nmap_results.txt; then #if service was found
                echo -e "${GC}$service found, checking for weak passwords...${NC}\n"

                PORT=${SERVICE_PORTS[$service]} #define PORT with the corrospnding one from the previously defined array

                # applying the selected brute-force method in the menu stage.
                case $METHOD in
                    1)
                        #custom username + custom password list
                        echo "$USERNAME" > /tmp/single_user.txt #temporary user-defined username
                        nmap -p"$PORT" --script "$service"-brute.nse --script-args userdb=/tmp/single_user.txt,passdb=$PASS_PATH $ip -oN $dir_name/$ip/${service}_brute.txt > /dev/null 2>&1
                        cat "$dir_name/$ip/${service}_brute.txt" >> $report_file
                        grep -i -B 4 "valid" "$dir_name/$ip/${service}_brute.txt" #grep only "valid" occurences + 4 lines prior to it.
                        echo ""
                        rm /tmp/single_user.txt #remove user-defined username
                        echo -e "${CC}$service scan for weak-passwords complete. The output was saved to $dir_name/$ip/${service}_brute.txt${NC}\n"
                        ;;
                    2)
                        #custom username + built-in password list
                        echo "$USERNAME" > /tmp/single_user.txt #temporary user-defined username
                        nmap -p"$PORT" --script "$service"-brute.nse --script-args userdb=/tmp/single_user.txt,passdb=./10k-most-common-passwords.txt $ip -oN $dir_name/$ip/${service}_brute.txt > /dev/null 2>&1
                        cat "$dir_name/$ip/${service}_brute.txt" >> $report_file
                        rm /tmp/single_user.txt #remove user-defined username
                        grep -i -B 4 "valid" "$dir_name/$ip/${service}_brute.txt" #grep only "valid" occurences + 4 lines prior to it.
                        echo ""
                        echo -e "${CC}$service scan for weak-passwords complete. The output was saved to $dir_name/$ip/${service}_brute.txt${NC}\n"
                        ;;
                    3)
                        #custom username list + custom password list
                        nmap -p"$PORT" --script "$service"-brute.nse --script-args userdb=$USER_PATH,passdb=$PASS_PATH $ip -oN $dir_name/$ip/${service}_brute.txt > /dev/null 2>&1
                        cat "$dir_name/$ip/${service}_brute.txt" >> $report_file
                        grep -i -B 4 "valid" "$dir_name/$ip/${service}_brute.txt" #grep only "valid" occurences + 4 lines prior to it.
                        echo ""
                        echo -e "${CC}$service scan for weak-passwords complete. The output was saved to $dir_name/$ip/${service}_brute.txt${NC}\n"
                        ;;
                    4)
                        #custom username list + built-in password list
                        nmap -p"$PORT" --script "$service"-brute.nse --script-args userdb=$USER_PATH,passdb=./10k-most-common-passwords.txt $ip -oN $dir_name/$ip/${service}_brute.txt > /dev/null 2>&1
                        cat "$dir_name/$ip/${service}_brute.txt" >> $report_file
                        grep -i -B 4 "valid" "$dir_name/$ip/${service}_brute.txt" #grep only "valid" occurences + 4 lines prior to it.
                        echo ""
                        echo -e "${CC}$service scan for weak-passwords complete. The output was saved to $dir_name/$ip/${service}_brute.txt${NC}\n"
                        ;;
                    5)
                        #built-in users&password lists
                        nmap -p"$PORT" --script "$service"-brute.nse --script-args userdb=top-usernames-shortlist.txt,passdb=./10k-most-common-passwords.txt $ip -oN $dir_name/$ip/${service}_brute.txt > /dev/null 2>&1
                        cat "$dir_name/$ip/${service}_brute.txt" >> $report_file
                        grep -i -B 4 "valid" "$dir_name/$ip/${service}_brute.txt" #grep only "valid" occurences + 4 lines prior to it.
                        echo ""
                        echo -e "${CC}$service scan for weak-passwords complete. The output was saved to $dir_name/$ip/${service}_brute.txt${NC}\n"
                        ;;
                esac
            else
                echo -e "${RC}$service not found.${NC}\n"
            fi
        done
}
```

Scanning TCP ports(nmap) -

This function uses previously defined variables to run an *"nmap"* scan when a host has been discovered, to map TCP services running on it and save the output to a separate text file and add it to the report.

```
#Function to scan for TCP ports
function NMAP (){
        echo -e "Using nmap to check for reachable TCP ports. Please wait..."
        nmap $ip -sV -oN $dir_name/$ip/nmap_results.txt > /dev/null 2>&1 #run nmap + versions and save to file
        echo -e "==== NMAP ====" >> "$report_file" #header
        cat $dir_name/$ip/nmap_results.txt >> $report_file #add to report
        echo -e "================\n" >> $report_file #footer
        echo -e "${CC}nmap complete. The output was saved to $dir_name/$ip/nmap_results.txt${NC}"
}
```

Scanning UDP ports(masscan) -

Similar to the "NMAP" function, this function tries to reach UDP services on the discovered host, and saves the found data to a separate text file while also adding to the report.

```
#Function to scan for UDP ports
function MASSCAN () {
        echo -e "\nUsing masscan to check for reachable UDP ports. Please wait..."
        masscan $ip -pU:1-65535 --rate 10000 2>/dev/null >> $dir_name/$ip/masscan_results.txt #run masscan and save to file
        echo -e "==== MASSCAN ====" >> "$report_file" #header
        cat $dir_name/$ip/masscan_results.txt >> $report_file #add to report
        echo -e "================\n" >> $report_file #footer
        echo -e "${CC}masscan complete. The output was saved to $dir_name/$ip/masscan_results.txt${NC}\n"
}
```

Service Vulnerability Analysis with SearchSploit -

This function executes some text manipulation to grab the services versions which are later used to be iterated over while using SearchSploit, to map known vulnerabilities.
Once done, some more text manipulation is in order to make the output not so busy with less important data.
Finally, the output gets saved to a separate text file and added to the report.

```
#Function to run searchsploit to find vulnerable services and save data.
function SPLOIT () {
        #awking only the version of each service from nmap + removing empty lines
        grep 'open' $dir_name/$ip/nmap_results.txt | awk '{ $1=$2=$3=""; print $0}' >> $dir_name/$ip/versions.txt
        sed -i '/^\s*$/d' $dir_name/$ip/versions.txt

        echo -e "Running searchsploit against the found services. Please wait..."
        echo -e "==== SEARCHSPLOIT ====" >> "$report_file" #header
        IFS=$'\n' #setting the internal field separator to split input by lines only.
        for service in $(cat $dir_name/$ip/versions.txt); do
    {
        echo "===== $service =====" #header
        #running searchsploit against every serivce version found earlier
        searchsploit "$service" | grep -v -E '(Exploits: No Results|Shellcodes: No Results)'
        echo ""
    } | sudo tee -a $dir_name/$ip/sploit_results.txt > /dev/null #writing output to file.

        done
        echo "SEARCHSPLOIT:"
        cat "$dir_name/$ip/sploit_results.txt"
        echo ""
        unset IFS #resets the internal field seperator to default.
        echo -e "${CC}searchsploit complete. The output was saved to $dir_name/$ip/sploit_results.txt${NC}\n"
        cat $dir_name/$ip/sploit_results.txt >> $report_file #add to report
        echo -e "================\n" >> $report_file #footer
}
```

Vulnerability Detection Using NSE Vulners Script(TCP) -

This function uses the Nmap Scripting Engine in order to find vulnerabilities for discovered TCP services.
It uses minor text manipulation to make the output a bit cleaner prior saving it to the relevant files.

```
#Function to run TCP .NSE vulners script on each found host
function VULNTCP () {

        echo "Searching for known TCP vulnerabilities. Please wait..."
        nmap $ip --script=vulners.nse -sV -oN "$dir_name/$ip/NSE_TCPvuln.txt" > /dev/null 2>&1 #run vulners script and save to file
        echo -e "==== NSE_vulnerabilities_TCP ====" >> "$report_file" #header
        awk '{print $1, $2, $3, $5}' "$dir_name/$ip/NSE_TCPvuln.txt" | tee -a "$report_file" #text-manipulation on the output + adding to report.
        echo -e "${CC}Vulnerabilites scan for TCP ports is complete. Results saved to $dir_name/$ip/NSE_TCPvuln.txt${NC}\n"
        echo -e "================\n" >> $report_file #footer
}
```

Vulnerability Detection Using NSE Vulners Script(UDP) -

Similar to the TCP version, this function uses the Nmap Scripting Engine in order to find vulnerabilities for discovered UDP services. In order to save time, instead of discovering open UDP ports once more, it works against the previously discovered ports by the "MASSCAN" function. For that, It utilizes the following command against the masscan results -

```
port=$(echo $line | grep -oP '\d+/udp' | cut -d'/' -f1)
```

This line searches for a number followed by */udp* in the line, then extracts only the number before the slash. For that, it uses "*grep -oP*" with a Perl-style regex (\d+) to match and show the part where the digits appear (the port number). After that, "*cut*" is used with a delimiter (*-d'/'*) option to treat the slash (/) as a separator and the *-f1* flag to keep only the first part of the result (the port number before the slash). The result is then saved into the *port* variable and used in the *"for loop"* for the NSE scan.

```
#Function to run UDP .NSE vulners script on each found host
function VULNUDP () {
    echo "Searching for known vulnerabilities on UDP ports. Please wait..."

    IFS=$'\n' #sets IFS to newline to split input correctly
    #loops through each line in masscan_results.txt
    for line in $(cat "$dir_name/$ip/masscan_results.txt"); do
        #extracts the UDP port number from each line
        port=$(echo "$line" | grep -oP '\d+/udp' | cut -d'/' -f1)

        #if a valid port is found, run Nmap for vulnerabilities
        if [[ ! -z "$port" ]]; then
            echo "Running Nmap for UDP port $port..."

            #runs the Nmap scan using the vulners script and save the results
            sudo nmap -sU -p $port -sV $ip --script=vulners >> "$dir_name/$ip/NSE_UDPvuln.txt" 2>&1
        fi
    done

    unset IFS #unsets IFS to restore default behavior

        #adds results to the report
        echo -e "==== NSE_vulnerabilities_UDP ====" >> "$report_file" #header
        awk '{print $1, $2, $3, $5}' "$dir_name/$ip/NSE_UDPvuln.txt" | tee -a "$report_file" #text-manipulation on the output + adding to report.
        echo -e "================\n" >> "$report_file" #footer

    echo -e "${CC}Vulnerability scan for UDP ports is complete. Results saved to $dir_name/$ip/NSE_UDPvuln.txt${NC}\n"
}
```

Searching the report -

This function provides the user the option to search through the generated report, once the scan is complete.
It uses three *"while true"* loops:

1. **First loop** - asks the user if they want to search for a specific output and will continue asking until the input is valid.
2. **Second loop –** In case the user chooses to search for a string in the first loop, this one will use *"grep -i"* to perform the search and move to the third loop.
3. **Third loop** – This loop handles the case in which the user is being asked if he wants to perform another search after already doing one previously. This loop is also looking for a valid input from the user.
   In case the input is "y", we move back to the 2nd loop and so on until the input for the 3rd loop is "n",at which point the *"return"* command is being used to exit the function.

Important notice - Without the 2nd loop, the user would only be able to search once.

```
#Function letting the user search for specific strings in the report file
function REPORT_SEARCH() {
    while true; do
    #ask if the user wants to perform a search
    echo -n "Do you want to search for a specific string in the found results? (y/n): "
    read search_choice

    case "$search_choice" in
    [Yy]*)
        #if user chooses 'y', ask for the string to search for and filter results
        while true; do
        echo -n "Enter the string to search for: "
        read search_string
        echo -e "Filtered results for '$search_string':\n"
        grep -i -n "$search_string" $dir_name/full_report.txt

        #ask if the user wants to search again
        while true; do
        echo -en "\nDo you want to search again? (y/n): "
        read search_again

        #check for valid input in additional searches (y/n)
        case "$search_again" in

        [Yy]*)
            #continue searching if 'y' is chosen
            break  # Break out of this inner loop, continue searching

        ;;

        [Nn]*)
            #break out and return to menu if 'n' is chosen
            echo -e "Exiting search."
            return

        ;;

        *)
            #if invalid input prompt
            echo -e "${RC}Invalid input. Please enter 'y' or 'n'.${NC}\n"

        ;;

        esac
        done
        done

        ;;

        [Nn]*)
                #if user chooses 'n' in the starting question, skip the search and return to the menu
                echo -e "Exiting search."
                return

        ;;

        *)
            #if invalid input for the starting question, ask again
            echo -e "${RED}Invalid input. Please enter 'y' or 'n'.${NC}\n"

        ;;

    esac
    done
}
```

Compressing all results into a ZIP file -

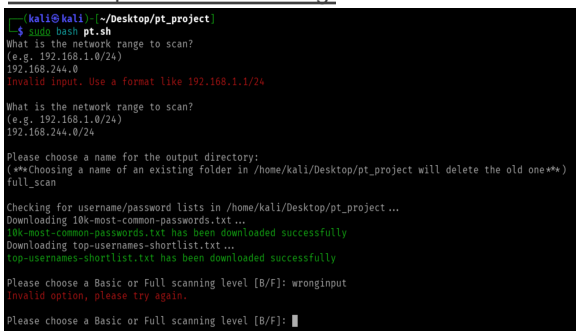
This function defines a timestamp to be added to the ZIP file name for two reasons.
1. Prevent from overwriting previous ZIP files that might exist in the same folder by giving it a unique name.
2. Making it easier for the user to find the desired ZIP by date and time of creation.

```
#Function to zip all result folder + report.txt.
function ZIP_RESULTS (){
        echo ""
        read -p "Would you like to save the results into a ZIP-file?(y/n)" ZIP_CHOICE

        case $ZIP_CHOICE in
            [Yy]*)
                echo -e "ZIPPING your files, please wait!"
                timestamp=$(date +%Y%m%d_%H%M%S) #current time
                zip -rq "$dir_name"_"$timestamp".zip $dir_name/ #zip the folder without terminal output
                echo -e "${CC}ZIP file was saved to $MAIN/$dir_name"_"$timestamp${NC}\n"
                echo -e "Exiting script. May the Force be with you."
                exit 1
            ;;

            [Nn]*)
                echo -e "Exiting script. May the Force be with you."
                exit 1
            ;;

            *)
                #If invalid input for the starting question, ask again
                echo -e "${RC}Invalid input. Please enter 'y' or 'n'.${NC}\n"
                ZIP_RESULTS

            ;;

        esac
}
```

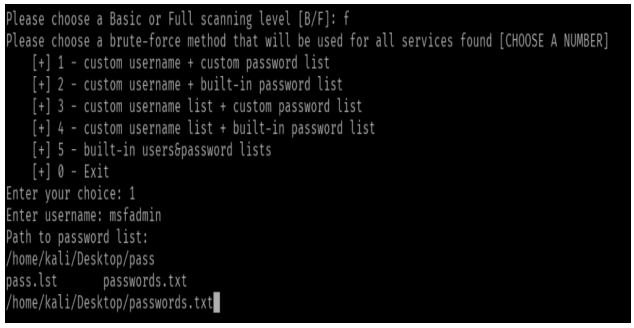
Demonstrating Script Performance -

This part will show **some** of the script functionality in work.
The following screenshots will demonstrate mostly the **Full Scanning** functionality in action. The **Basic Scanning** feature operates similarly but provides a more limited output, as it does not include vulnerability mappings.

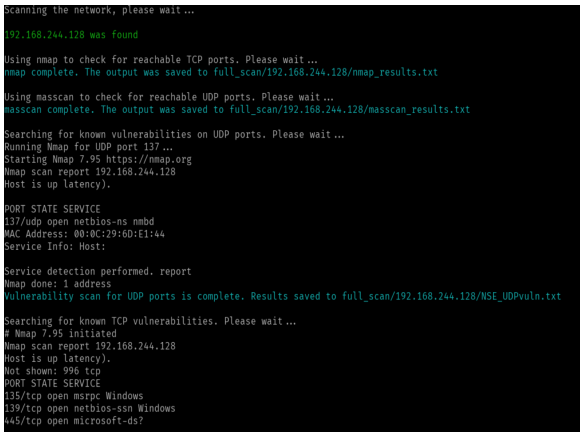
Initial setup and error handling:



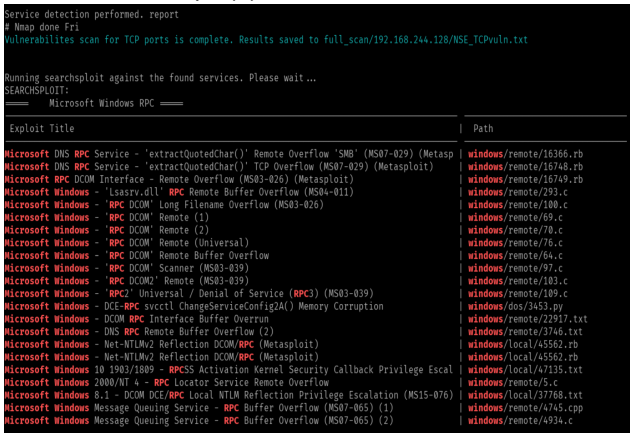
Scanning level + choosing a method:



Windows host output(1):



Windows host output(2):



Windows host output(3):

```
Exploit Title                                                    |  Path
Microsoft Terminal Services - Use-After-Free (MS12-020)          | windows/dos/18606.txt

searchsploit complete. The output was saved to full_scan/192.168.244.128/sploit_results.txt

Checking for weak passwords against common services. Please wait ...

ssh not found.

ftp not found.

telnet not found.

rdp not found.

192.168.244.133 was found
```

Metasploitable host output(1):

```
192.168.244.133 was found

Using nmap to check for reachable TCP ports. Please wait ...
nmap complete. The output was saved to full_scan/192.168.244.133/nmap_results.txt

Using masscan to check for reachable UDP ports. Please wait ...
masscan complete. The output was saved to full_scan/192.168.244.133/masscan_results.txt

Searching for known vulnerabilities on UDP ports. Please wait ...
Running Nmap for UDP port 53 ...
Running Nmap for UDP port 137 ...
Starting Nmap 7.95 https://nmap.org
Nmap scan report 192.168.244.133
Host is up latency).

PORT STATE SERVICE
53/udp open domain BIND
| vulners:
| cpe:/a:isc:bind:9.4.2:
| SSV:2853 10.0 *EXPLOIT*
| FF02A46B-33EF-5995-B22B-451D42342473 10.0 *EXPLOIT*
| CVE-2008-0122 10.0
| CVE-2021-25216 9.8
| CVE-2020-8616 8.6
| CVE-2016-1286 8.6
| SSV:60184 8.5 *EXPLOIT*
| CVE-2012-1667 8.5
| SSV:60292 7.8 *EXPLOIT*
| PACKETSTORM:180552 7.8 *EXPLOIT*
| PACKETSTORM:180551 7.8 *EXPLOIT*
| PACKETSTORM:138960 7.8 *EXPLOIT*
| PACKETSTORM:132926 7.8 *EXPLOIT*
| MSF:AUXILIARY-DOS-DNS-BIND_TKEY- 7.8 *EXPLOIT*
```

Metasploitable host output(2):

```
| SSV:14986 2.6 *EXPLOIT*
| CVE-2009-4022 2.6
| PACKETSTORM:142800 0.0 *EXPLOIT*
|_ 1337DAY-ID-27896 0.0 *EXPLOIT*
MAC Address: 00:0C:29:28:88:05

Service detection performed. report
Nmap done: 1 address
Starting Nmap 7.95 https://nmap.org
Nmap scan report 192.168.244.133
Host is up latency).

PORT STATE SERVICE
137/udp open netbios-ns Windows
MAC Address: 00:0C:29:28:88:05
Service Info: Host: OS:

Service detection performed. report
Nmap done: 1 address
Vulnerability scan for UDP ports is complete. Results saved to full_scan/192.168.244.133/NSE_UDPvuln.txt

Searching for known TCP vulnerabilities. Please wait ...
# Nmap 7.95 initiated
Nmap scan report 192.168.244.133
Host is up latency).
Not shown: 979 tcp
PORT STATE SERVICE
21/tcp open ftp 2.3.4
| vulners:
| vsftpd 2.3.4:
| PACKETSTORM:162145 10.0 *EXPLOIT*
| FF02A46B-33EF-5995-B22B-451D42342473 10.0 *EXPLOIT*
| EDB-ID:49757 10.0 *EXPLOIT*
| CVE-2011-2523 10.0
```

Metaspolitable host output(3):

```
| SSV:19322 3.5 *EXPLOIT*
| POSTGRESQL:CVE-2019-10209 3.5
| PACKETSTORM:127092 3.5 *EXPLOIT*
|_ CVE-2010-0733 3.5
6667/tcp open irc
8009/tcp open ajp13 Jserv
8180/tcp open http Tomcat/Coyote
|_http-server-header: Apache-Coyote/1.1
MAC Address: 00:0C:29:28:88:05
Service Info: Hosts: irc.Metasploitable.LAN;

Service detection performed. report
# Nmap done Fri
Vulnerabilites scan for TCP ports is complete. Results saved to full_scan/192.168.244.133/NSE_TCPvuln.txt


Running searchsploit against the found services. Please wait ...
SEARCHSPLOIT:
===   vsftpd 2.3.4 ===

Exploit Title                                                    |  Path
vsftpd 2.3.4 - Backdoor Command Execution                        | unix/remote/49757.py
vsftpd 2.3.4 - Backdoor Command Execution (Metasploit)           | unix/remote/17491.rb

===   OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0) ===

===   Linux telnetd ===

Exploit Title                                                    |  Path
netkit-telnet-0.17 telnetd (Fedora 31) - 'BraveStarr' Remote Code Execution | linux/remote/48170.py
TelnetD encrypt_keyid - Function Pointer Overwrite               | linux/remote/18280.c
```

Metasploitable output(4):

```
searchsploit complete. The output was saved to full_scan/192.168.244.133/sploit_results.txt

Checking for weak passwords against common services. Please wait ...

ssh found, checking for weak passwords ...

PORT   STATE SERVICE
22/tcp open  ssh
| ssh-brute:
|   Accounts:
|     msfadmin:msfadmin - Valid credentials

ssh scan for weak-passwords complete. The output was saved to full_scan/192.168.244.133/ssh_brute.txt

ftp found, checking for weak passwords ...

PORT   STATE SERVICE
21/tcp open  ftp
| ftp-brute:
|   Accounts:
|     msfadmin:msfadmin - Valid credentials

ftp scan for weak-passwords complete. The output was saved to full_scan/192.168.244.133/ftp_brute.txt

telnet found, checking for weak passwords ...

PORT   STATE SERVICE
23/tcp open  telnet
| telnet-brute:
|   Accounts:
|     msfadmin:msfadmin - Valid credentials

telnet scan for weak-passwords complete. The output was saved to full_scan/192.168.244.133/telnet_brute.txt
```

Full report start:

```
23/tcp open  telnet
| telnet-brute:
|   Accounts:
|     msfadmin:msfadmin - Valid credentials

telnet scan for weak-passwords complete. The output was saved to full_scan/192.168.244.133/telnet_brute.txt

rdp not found.

Showing full report:

     1  === 192.168.244.128 ===
     2  === NMAP ===
     3  # Nmap 7.95 scan initiated Fri Aug 22 10:16:13 2025 as: /usr/lib/nmap/nmap -sV -oN full_scan/192.168.244.128/nmap_res
ults.txt 192.168.244.128
     4  Nmap scan report for 192.168.244.128
     5  Host is up (0.00028s latency).
     6  Not shown: 996 closed tcp ports (reset)
     7  PORT     STATE SERVICE       VERSION
     8  135/tcp  open  msrpc         Microsoft Windows RPC
     9  139/tcp  open  netbios-ssn   Microsoft Windows netbios-ssn
    10  445/tcp  open  microsoft-ds?
    11  3389/tcp open  ms-wbt-server Microsoft Terminal Services
    12  MAC Address: 00:0C:29:6D:E1:44 (VMware)
    13  Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
    14
    15  Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
    16  # Nmap done at Fri Aug 22 10:16:23 2025 -- 1 IP address (1 host up) scanned in 9.36 seconds
    17  ==============
    18
    19  === MASSCAN ===
    20  Discovered open port 137/udp on 192.168.244.128
    21  ==============
    22
```

Full report end:

```
1399  # Nmap 7.95 scan initiated Fri Aug 22 10:17:57 2025 as: /usr/lib/nmap/nmap -p21 --script ftp-brute.nse --script-args
userdb=/tmp/single_user.txt,passdb=/home/kali/Desktop/passwords.txt -oN full_scan/192.168.244.133/ftp_brute.txt 192.168.244.1
33
1400  Nmap scan report for 192.168.244.133
1401  Host is up (0.00049s latency).
1402
1403  PORT   STATE SERVICE
1404  21/tcp open  ftp
1405  | ftp-brute:
1406  |   Accounts:
1407  |     msfadmin:msfadmin - Valid credentials
1408  |_  Statistics: Performed 5 guesses in 3 seconds, average tps: 1.7
1409  MAC Address: 00:0C:29:28:88:05 (VMware)
1410
1411  # Nmap done at Fri Aug 22 10:18:01 2025 -- 1 IP address (1 host up) scanned in 3.33 seconds
1412  # Nmap 7.95 scan initiated Fri Aug 22 10:18:01 2025 as: /usr/lib/nmap/nmap -p23 --script telnet-brute.nse --script-ar
gs userdb=/tmp/single_user.txt,passdb=/home/kali/Desktop/passwords.txt -oN full_scan/192.168.244.133/telnet_brute.txt 192.168
.244.133
1413  Nmap scan report for 192.168.244.133
1414  Host is up (0.00048s latency).
1415
1416  PORT   STATE SERVICE
1417  23/tcp open  telnet
1418  | telnet-brute:
1419  |   Accounts:
1420  |     msfadmin:msfadmin - Valid credentials
1421  |_  Statistics: Performed 5 guesses in 3 seconds, average tps: 1.7
1422  MAC Address: 00:0C:29:28:88:05 (VMware)
1423
1424  # Nmap done at Fri Aug 22 10:18:04 2025 -- 1 IP address (1 host up) scanned in 2.84 seconds

This output was saved to full_scan/full_report.txt
Do you want to search for a specific string in the found results? (y/n):
```

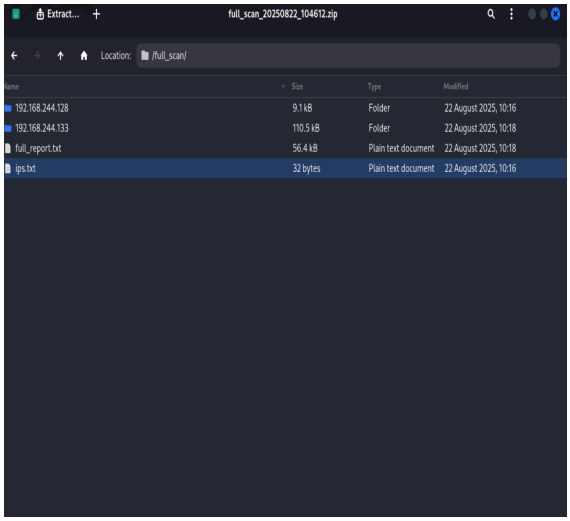
Full report searching + error handling:

```
Do you want to search for a specific string in the found results? (y/n):
wronginput
Invalid input. Please enter 'y' or 'n'.

Do you want to search for a specific string in the found results? (y/n):
y
Enter the string to search for:
valid
Filtered results for 'valid':

1394:|     msfadmin:msfadmin - Valid credentials
1407:|     msfadmin:msfadmin - Valid credentials
1420:|     msfadmin:msfadmin - Valid credentials

Do you want to search again? (y/n):
wronginput
Invalid input. Please enter 'y' or 'n'.


Do you want to search again? (y/n):
```

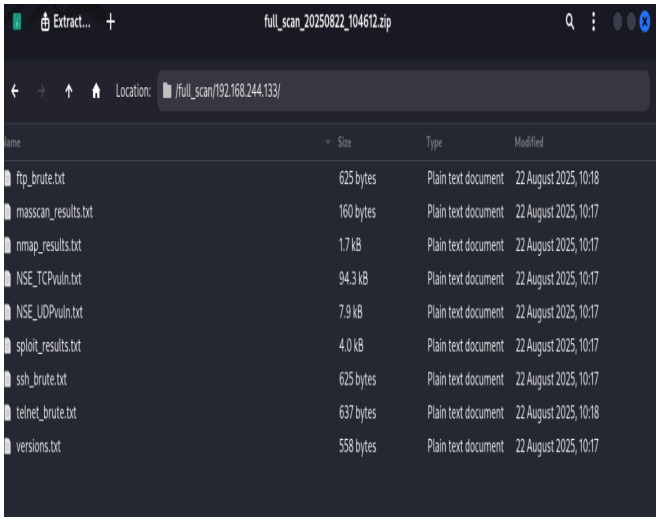
Full report generation + zipping folder with a timestamp:

```
Do you want to search again? (y/n):
n
Exiting search.

Would you like to save the results into a ZIP-file?(y/n)y
ZIPPING your files, please wait!
ZIP file was saved to /home/kali/Desktop/pt_project/full_scan_20250822_104612

Exiting script. May the Force be with you.
```

Zip Content(1):



Zip Content(2):



full_report.txt file example(1):

```
1 == 192.168.244.128 ==
2 == NMAP ==
3 # Nmap 7.95 scan initiated Fri Aug 22 10:16:13 2025 as: /usr/lib/nmap/nmap -sV -oN full_scan/192.168.244.128/nmap_results.txt 192.168.244.128
4 Nmap scan report for 192.168.244.128
5 Host is up (0.00028s latency).
6 Not shown: 996 closed tcp ports (reset)
7 PORT    STATE SERVICE      VERSION
8 135/tcp  open  msrpc        Microsoft Windows RPC
9 139/tcp  open  netbios-ssn  Microsoft Windows netbios-ssn
10 445/tcp  open  microsoft-ds?
11 3389/tcp open  ms-wbt-server Microsoft Terminal Services
12 MAC Address: 00:0C:29:6D:E1:44 (VMware)
13 Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
14
15 Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
16 # Nmap done at Fri Aug 22 10:16:23 2025 -- 1 IP address (1 host up) scanned in 9.36 seconds
17 ==========
18
19 == MASSCAN ==
20 Discovered open port 137/udp on 192.168.244.128
21 ==========
22
23 == NSE_vulnerabilities_UDP ==
24 Starting Nmap 7.95 https://nmap.org
25 Nmap scan report 192.168.244.128
26 Host is up latency).
27
28 PORT STATE SERVICE
29 137/udp open netbios-ns nmbd
30 MAC Address: 00:0C:29:6D:E1:44
31 Service Info: Host:
32
33 Service detection performed. report
```

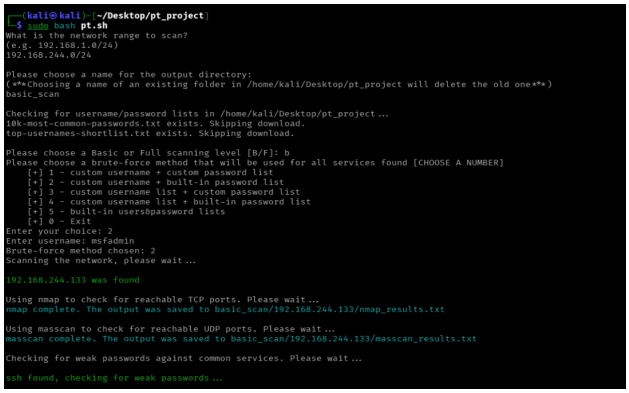
full_report.txt file example(2):

```
104 == 192.168.244.133 ==
105 == NMAP ==
106 # Nmap 7.95 scan initiated Fri Aug 22 10:16:59 2025 as: /usr/lib/nmap/nmap -sV -oN full_scan/192.168.244.133/nmap_results.txt 192.168.244.133
107 Nmap scan report for 192.168.244.133
108 Host is up (0.0035s latency).
109 Not shown: 979 closed tcp ports (reset)
110 PORT    STATE SERVICE     VERSION
111 21/tcp  open  ftp         vsftpd 2.3.4
112 22/tcp  open  ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
113 23/tcp  open  telnet      Linux telnetd
114 25/tcp  open  smtp        Postfix smtpd
115 53/tcp  open  domain      ISC BIND 9.4.2
116 80/tcp  open  http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
117 111/tcp  open  rpcbind     2 (RPC #100000)
118 139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
119 445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
120 512/tcp  open  exec        netkit-rsh rexecd
121 513/tcp  open  login       OpenBSD or Solaris rlogind
122 514/tcp  open  tcpwrapped
123 1099/tcp open  java-rmi    GNU Classpath grmiregistry
124 1524/tcp open  bindshell   Metasploitable root shell
125 2049/tcp open  nfs         2-4 (RPC #100003)
126 2121/tcp open  ftp         ProFTPD 1.3.1
127 3306/tcp open  mysql       MySQL 5.0.51a-3ubuntu5
128 5432/tcp open  postgresql  PostgreSQL DB 8.3.0 - 8.3.7
129 6667/tcp open  irc         UnrealIRCd
130 8009/tcp open  ajp13       Apache Jserv (Protocol v1.3)
131 8180/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
132 MAC Address: 00:0C:29:28:88:05 (VMware)
```

full_report.txt file example(3):

```
1370 ─────────────────────────────────────────────────────────────────────────────── ──────────────────────────────────
1371  Exploit Title                                                                |  Path
1372 ─────────────────────────────────────────────────────────────────────────────── ──────────────────────────────────
1373 [01;31m[KUnrealIRCd[m[K 3.2.8.1 - Backdoor Command Execution (Metasploit)                    | linux/remote/16922.rb
1374 [01;31m[KUnrealIRCd[m[K 3.2.8.1 - Local Configuration Stack Overflow                        | windows/dos/18011.txt
1375 [01;31m[KUnrealIRCd[m[K 3.2.8.1 - Remote Downloader/Execute                               | linux/remote/13853.pl
1376 [01;31m[KUnrealIRCd[m[K 3.x - Remote Denial of Service                                    | windows/dos/27407.pl
1377 ─────────────────────────────────────────────────────────────────────────────── ──────────────────────────────────
1378
1379 =====    Apache Jserv (Protocol v1.3) =====
1380
1381 =====    Apache Tomcat/Coyote JSP engine 1.1 =====
1382
1383 ==========
1384
1385 ==== BRUTE_NSE ====
1386 # Nmap 7.95 scan initiated Fri Aug 22 10:17:55 2025 as: /usr/lib/nmap/nmap -p22 --script ssh-brute.nse --script-args userdb=/tmp/
     single_user.txt,passdb=/home/kali/Desktop/passwords.txt -oN full_scan/192.168.244.133/ssh_brute.txt 192.168.244.133
1387 Nmap scan report for 192.168.244.133
1388 Host is up (0.00049s latency).
1389
1390 PORT   STATE SERVICE
1391 22/tcp open  ssh
1392 | ssh-brute:
1393 |   Accounts:
1394 |     msfadmin:msfadmin - Valid credentials
1395 |_  Statistics: Performed 5 guesses in 2 seconds, average tps: 2.5
1396 MAC Address: 00:0C:29:28:88:05 (VMware)
1397
```

BASIC(1) - basic scan - credential files exist+method 2



BASIC(2) - basic scan - skipping vulnerabilities mapping