Linux Fundamentals | Project 3: Network Research

Muhammad Termidzi Bin Azmi (S27)

CFC190324

Samson Xiao

27 May 2024

TABLE OF CONTENTS

INTRODUCTION

This project involves the creation of an automated script for secure and anonymous network interactions via remote servers to launch an attack on target domain or IP address (with proper authorization). The script checks for necessary application installations, verifies network anonymity, and allows the user to specify an address for remote server scanning. Upon confirming network anonimity, the script proceeds to connect to the remote server via SSH.

After connecting, the script displays the server's details, performs a Whois check, and scans for open ports on the given address. The collected Whois and Nmap data are then saved onto the remote server and are later collected via ftp into the local machines. Finally, a log is created for auditing the data collection process. This automated script is a comprehensive tool for network security and data collection, ensuring secure and anonymous network connections while performing detailed network scans.

This report aims to create a detailed analysis of Operation Remote Infiltration, where it centers around a straightforward goal: Running an automated script from local device which would be executed by the remote server for inteligence gathering.

Important Note: Ethical hacking are critical activities for maintaining robust network security. However, they require proper authorization and strict adherence to legal frameworks. This script should only be used in controlled environments with explicit permission from the system owner.

METHODOLOGIES

SCOPE 1: NETWORK REMOTE CONTROL

1. **Check to see if needed applications are installed, and to install them.**

```
23  function INSTALL()
24  {
25      # check status. how? version?
26      echo "-----------------------------------------"
27      echo
28      echo "Checking to see if required tools are installed:"
29      sleep 1
30      INSTALL_NIPE
31      sleep 1
32      INSTALL_GIT
33      sleep 1
34      INSTALL_GEOIP-BIN
35      sleep 1
36      INSTALL_NMAP
37      sleep 1
38      INSTALL_SSHPASS
39      sleep 1
40      INSTALL_TOR
41      sleep 1
42      echo
43      echo "All required tools good to go!"
44      echo
45      echo "-----------------------------------------"
46      sleep 1
47  }
```

Methods used here:

- **(line 23) function**: In Linux, a function is a way to group commands for later use, making it easier to perform repetitive tasks. For example, INSTALL, INSTALL_NIPE, INSTALL_GIT
- **(line 26) echo**: It is used to display lines of text or strings that are passed as arguments. It's a powerful tool for outputting status text to the screen or a file in shell scripts and batch files.
- **(line 29) sleep**: This command delays the execution of subsequent commands for a specified amount of time. The default unit is seconds, but you can also specify minutes (m), hours (h), or days (d).

2. Installing Nipe

```
51  function INSTALL_NIPE()
52  {
53
54      NIPE_DIR=$(sudo find / -type d -name "nipe" 2>/dev/null)    # Search for 'nipe' directory
55      if [ -n "$NIPE_DIR" ]; then                                 #check for NIPE, if installed, echo 'NIPE INSTALLED', EXIT.
56          echo "[#] NIPE is already installed."
57          sleep 1
58      else
59          echo "NIPE is not installed. Installing now..."
60
61          git clone https://github.com/htrgouvea/nipe             # clone NIPE from git repo
62          cd nipe                                                 # cd in NIPE folder
63          cpanm --installdeps .                                   # install dependencies
64          sudo perl nipe.pl install                               # install NIPE
65          echo "[*] NIPE successfully installed."
66          sleep 1
67          INSTALL
68      fi
69  }
```

Methods used here:

- **(line 54) NIPE_DIR:** This is a variable in which the output of the command enclosed within the parentheses $(...) will be stored. The variable name can be configured accordingly.
- **(line 54) $(...):** This is called command substitution. It allows the output of a command to replace the command itself. The shell executes the command inside the $() and then replaces the entire $() construct with the output of the command (variable name).
- **(line 54) sudo:** This is a command that allows you to run programs with the security privileges of another user, by default the superuser (root). It's used here to ensure that the find command has the necessary permissions to search all directories in the file system
- **(line 54) find / -type d -name "nipe":** This is the find command that searches for files and directories in the file system.
    - **/ :** specifies the root directory as the starting point for the search.
    - **-type d:** restricts the search to directories only.
    - **-name "":** searches for a directory with the exact name in "".
- **(line 54) 2>/dev/null:** This part is about redirecting standard error.
    - **2:** represents the file descriptor for standard error.
    - **>:** redirection operator.
    - **/dev/null:** a special file that discards all data written to it (it's like a black hole for data).
    - Therefore, **2>/dev/null** redirects any error messages produced by the find command to /dev/null, effectively silencing any errors**.**
- **(line 55) if [-n "$NIPE_DIR"]; then:** This is the start of an if statement.
    - **if:** Begins the conditional statement.
    - **[-n "$NIPE_DIR"]:** The square brackets denote a test is being performed. -n tests if the string inside the variable NIPE_DIR is non-empty.
    - **then:** If the test returns true (meaning NIPE_DIR contains a path), the commands following then are executed.
- **(line 58) else:** This keyword is used to specify the alternative block of commands that should be executed if the test ([-n "$NIPE_DIR"]) returns false.

- **(line 61) git clone https://github.com/htrgouvea/nipe:** This command clones the NIPE repository from GitHub to the local machine.
    - **git:** This is the tool being used. Git is a version control system for tracking changes in computer files and coordinating work on those files among multiple people.
    - **clone:** This is a Git command that creates a copy of an existing repository. It's used to download the repository's content, including all of its history, into a new directory on your local machine.
    - **https://github.com/htrgouvea/nipe:** This is the URL of the remote repository you want to clone. It points to the location on GitHub where the NIPE repository is hosted. This can be replace with any other URL.
- **(line 62) cd nipe:** Changes the directory to the newly cloned nipe directory.
- **(line 63) cpanm --installdeps .:** Uses the cpanm (CPAN Minus) tool to install all Perl dependencies listed in the current directory (denoted by " . ").
- **(line 64) sudo perl nipe.pl install:** Runs the NIPE installation script with superuser privileges.
    - **sudo:** This is a command that allows users to run programs with the security privileges of the superuser or another user, which are often required for installation processes.
    - **perl:** This is the command-line interpreter for the Perl programming language.
    - **nipe.pl:** This refers to the Perl script file that is being executed. The .pl extension indicates that it is a Perl script. In this context, nipe.pl is the main script of the NIPE tool, which is a script to make Tor Network our default gateway.
    - **install:** This is an argument passed to the nipe.pl script. It tells the script to perform its installation routine.
- **(line 68) fi:** Ends the if statement block.

3. Nmap installation

```
79          sudo apt-get update
80          sudo apt-get install nmap -y
```

Methods used here:

- **(line 79) sudo apt-get update:** This command updates the list of available packages and their versions, but it does not install or upgrade any packages.
    - **sudo:** Runs the command as the superuser (root), allowing it to access protected system files.
    - **apt-get:** This is the command-line tool for handling packages.
    - **update:** This option tells apt-get to synchronize the package index files from their sources specified in /etc/apt/sources.list.
- **(line 80) sudo apt-get install nmap -y:** This command installs the nmap package.
    - **install**: This option tells apt-get to install a package.
    - **-y**: This flag automatically answers "yes" to prompts during the installation process, allowing the installation to proceed without manual intervention.

4. **Anonymity check function.**

```
154     IP_ADD=$(curl -s ifconfig.io)
155     if [ "$(geoiplookup $IP_ADD | grep -i SG)" ]
```

Methods used here:

- **(line 154) IP_ADD=$(curl -s ifconfig.io)**
  - **curl:** a tool for transferring data from or to a server, using one of the supported protocols (HTTP, HTTPS, FTP, and more). In this case, it's used to make a request to a web service.
  - **-s:** This is an option for curl that tells it to operate in "silent" or "quiet" mode.
  - **ifconfig.io:** This is the URL of a web service that returns the public IP address of the client that makes the request.
- **(line 155) $(geoiplookup $IP_ADD | grep -i SG)**
- **geoiplookup:** Takes an IP address as input and returns the geographical location information associated with it, such as the country, city, and other details.
- **| :** Pipe, which takes the output of the command on its left and uses it as input for the command on its right.
- **grep:** This is a grep command, which searches for a specific pattern in the input it receives.
- **-i:** This option makes the search case-insensitive.

5. **Nipe function status/start/stop**

```
181     sudo perl nipe.pl status
182     sleep 1
183     echo "Launch Status False. Starting NIPE."
184     sudo perl nipe.pl start
```

Methods used here:

- **(line 181) status:** This is an argument passed to the nipe.pl script, instructing it to report the current status of the NIPE system.
- **(line 184) start:** This is an argument passed to the nipe.pl script, instructing it to start the NIPE system.

6. **Allow the user to specify the address/URL and save into a variable**

```
196     read -p "[?] Specify a Domain/IP Address to Attack: " target
```

Methods used here:

- **(line 196) read -p "[?] Specify a Domain/IP Address to Attack: " target**
  - **read:** This is a shell built-in command that reads a line from the standard input.
  - **-p**: This option allows you to specify a prompt that is displayed to the user. The prompt in this case is "[?] Specify a Domain/IP Address to Attack: ".

- **target**: This is the name of the variable that will store the input from the user, can be changed accordingly.

7. **Scanning of remote server**

```
208     nmap 192.168.254.129
```

Methods used here:

- **(line 208) nmap:** This is the command for running Nmap, which is an open-source tool used for network exploration or security auditing.

8. **Connect remote and display uptime, IP address and country.**

```
229 function CONNECT_REMOTE()
230 {
231
232     export SSHPASS=''
233     echo
234     echo "[*] Establishing connection to Remote Server..."
235     echo
236     sleep 1
237     RMT_UPTIME          #get the uptime
238     sleep 1
239     RMT_IP              #get the IP Address
240     sleep 1
241     RMT_COUNTRY         #get the Country
242     echo
243     echo "-------------------------------------------"
244     sleep 1
245     echo
246 }
247
248 function RMT_UPTIME()
249 {
250     rmt_up=$(sshpass -e ssh -q tc@192.168.254.129 'uptime')
251     echo "Uptime: $rmt_up"
252 }
253
254 function RMT_IP()
255 {
256     rmt_ip=$(sshpass -e ssh -q tc@192.168.254.129 'curl -s ifconfig.io')
257     echo "IP Address: $rmt_ip"
258 }
259
260 function RMT_COUNTRY()
261 {
262     rmt_country=$(sshpass -e ssh -q tc@192.168.254.129 "geoiplookup $rmt_ip | cut -d ',' -f2 | cut -d ' ' -f2")
263     echo "Country: $rmt_country"
264
265 }
```

Methods used here:

- **(line 232) export SSHPASS='.'**
    - **export**: This keyword is used to make a variable accessible throughout the current shell session and any child processes it spawns.
    - **SSHPASS**: This is the name of the environment variable being created.
    - **='.'**: This assigns an empty string value to the variable. Here we put the SSH server password.

- **(line 250) sshpass -e ssh -q tc@192.168.254.129 'uptime'**:
    - **sshpass**: This utility is used to provide a password to the ssh command non-interactively. It allows you to automate SSH connections by passing the password directly as an argument.
    - **-e**: This flag specifies that the password should be read from the environment variable SSHPASS.
    - **ssh**: The SSH command-line tool for securely connecting to remote servers. It establishes an encrypted communication channel between the local and remote machines.
    - **-q**: This flag stands for "quiet" mode, which suppresses warning and diagnostic messages during the SSH session.
    - **tc@192.168.254.129**: This is the username (tc) and the IP address (192.168.254.129) of the remote server you want to connect to.
    - **'uptime'**: The single-quoted argument specifies the command you want to execute on the remote server. In this case, it's the uptime command, which displays the system uptime.
- **(line 250) "geoiplookup $rmt_ip | cut -d ',' -f2 | cut -d ' ' -f2"**: Getting the country code based on the IP address of the remote server. The double-quoted argument specifies the command we want to execute on the remote server.
    - **geoiplookup $rmt_ip:** Executes the geoiplookup command on the remote server, which provides information about the geographical location of an IP address.
    - **cut -d ',' -f2:** The first cut command extracts the second field (separated by commas) from the output of geoiplookup. This field typically contains the country information
    - **cut -d ' ' -f2:** The second cut command further extracts the second field (separated by spaces) from the previous result. This field usually represents the country code (e.g., "US").
        - **cut:** The cut command is used to extract specific sections (fields) from lines of text or files.
        - **-d ',':** This option specifies the delimiter character. In this case, the delimiter is a comma (,).
        - **-f2:** This option specifies the field(s) to extract. Here, it indicates that we want the second field (based on the delimiter) from each line.

9. Commencing attack

```
270     sudo touch /var/log/nr.log
```

Methods used here:

- **(line 270) touch:** Creates new, empty files. If provided a filename as an argument, touch will create an empty file with that name.

10. Whois target

```
299     sshpass -e ssh -q tc@192.168.254.129 "whois '$target' > whois_output.txt"
300     sleep 1
301     wget -q ftp://tc:tc@192.168.254.129/whois_output.txt
302     echo "[@] WHOIS data of $target was saved into $(pwd)"
303     sudo echo "$(date)- [*] WHOIS data collected for: $target" >> nr.log
```

Methods used here:

- **(line 299) "whois '$target' > whois_output.txt":** The double quotes establish this command via SSH connection to the remote server.
    - **whois '$target':** Executes the whois command with the value of the variable $target. The **whois** command provides information about domain names, IP addresses, and network-related details**.**
    - **>:** Redirects the output of the whois command to a file.
    - **whois_output.txt:** The filename where the output will be saved. This can be changed.
- **(line 301) wget -q ftp://tc:tc@192.168.254.129/whois_output.txt:**
    - **wget**: The wget command is a powerful utility for downloading files from the web. It supports various protocols, including HTTP, HTTPS, and FTP.
    - **-q**: This option stands for “quiet” mode. When used, wget suppresses output messages, making it suitable for automation or scripting.
    - **ftp://tc:tc@192.168.254.129/whois_output.txt**: This part specifies the URL from which to download the file
        - **ftp://:** File Transfer Protocol is commonly used for transferring files between a client (local machine) and a server (the specified IP address).
        - **tc:tc:** Represents the username and password used for authentication. Here both username:password is ‘tc’.
        - **/whois_output.txt**: Specifies the path to the file on the FTP server that we want to retrieve the file from.
- **(line 303) sudo echo "$(date)- [*] NMAP data collected for: $target" >> nr.log**
    - **date**: Display and set the system date and time.
    - **>> :** Operator is used for appending output to an existing file.

DISCUSSION

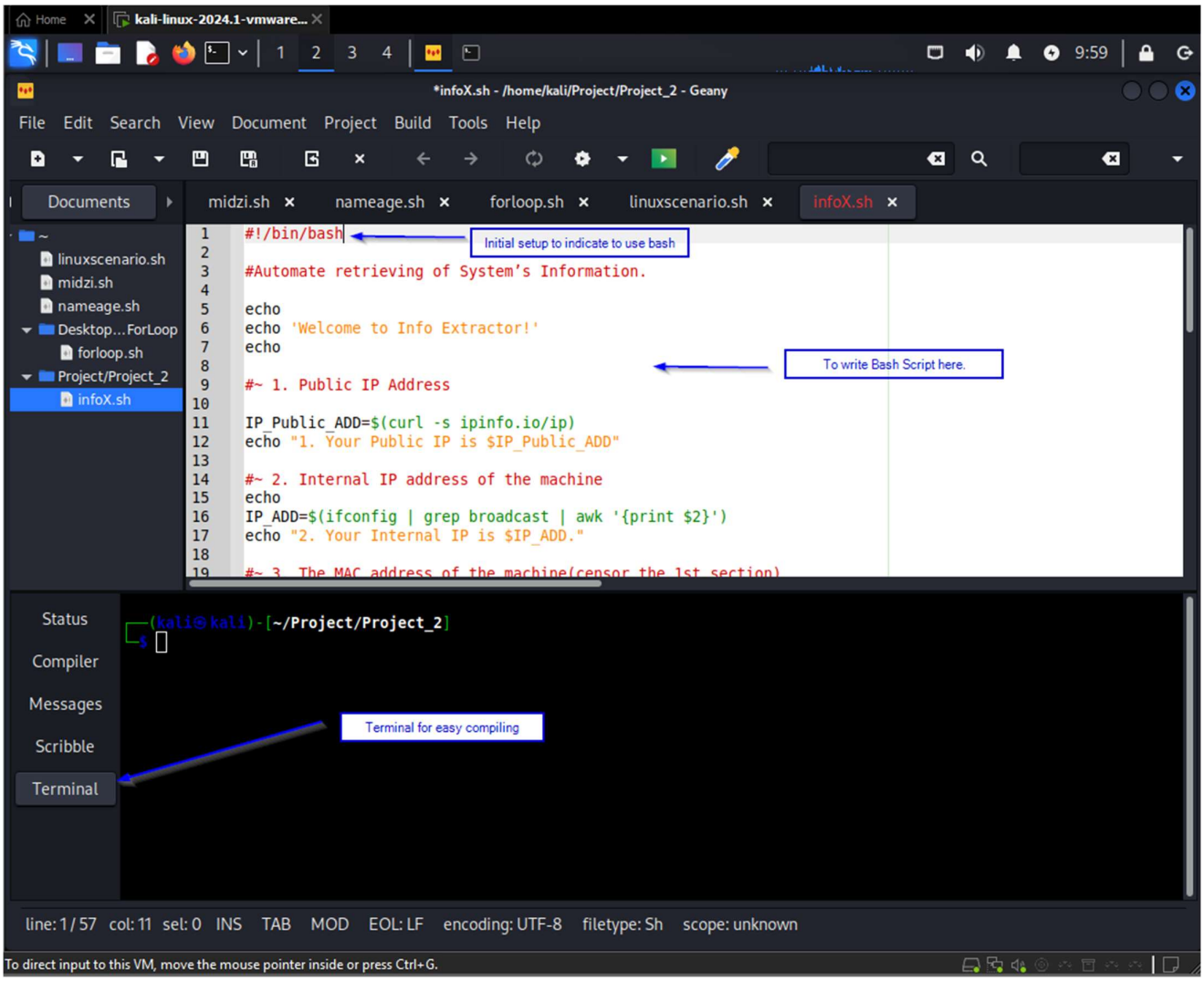
SCOPE 1: NETWORK REMOTE CONTROL

1. **Setting up Geany IDE**

- Using Geany - A free and open-source lightweight GUI text editor, including basic IDE feature.
- Download geany.
- Go into the intended folder to store the file.
- Type in “geany” into the linux command prompt followed by file name ending with .sh.

```
┌──(kali㉿kali)-[~/Project/Project_2]
└─$ geany infoX.sh
```

2. **Geany IDE navigation**



- Run with [**bash network research.sh**] on terminal.

3. Installations Check

```
23  function INSTALL()
24  {
25      # check status. how? version?
26      echo "-----------------------------------------"
27      echo
28      echo "Checking to see if required tools are installed:"
29      sleep 1
30      INSTALL_NIPE
31      sleep 1
32      INSTALL_GIT
33      sleep 1
34      INSTALL_GEOIP-BIN
35      sleep 1
36      INSTALL_NMAP
37      sleep 1
38      INSTALL_SSHPASS
39      sleep 1
40      INSTALL_TOR
41      sleep 1
42      echo
43      echo "All required tools good to go!"
44      echo
45      echo "-----------------------------------------"
46      sleep 1
47  }
```

- Install the needed applications.
- If the applications are already installed. If installed, don't install them again.
- In this function, installation functions (INSTALL_NIPE, INSTALL_GIT, etc.) are being called upon to check if the program is installed.
- Except for NIPE, the rest of the programs are checked with their version. This is an example of the installations check:

```
71  function INSTALL_NMAP()
72  {
73      NMAP_V=$(nmap --version)    #check for NMAP version
74      if [ -n "$NMAP_V" ]; then
75          echo "[#] NMAP is already installed."
76          sleep 1
77      else
78          "NMAP is not installed. Installing now..."
79          sudo apt-get update
80          sudo apt-get install nmap -y
81          echo "[*] NMAP successfully installed."
82          INSTALL
83      fi
84  }
```

- Conditional statements are being utilized. We will then update the database and install the required program.
- Here we can see the function for INSTALL_NIPE, where it searches for whether NIPE directory exists or not:

```
function INSTALL_NIPE()
{

    NIPE_DIR=$(sudo find / -type d -name "nipe" 2>/dev/null)    # Search for 'nipe' directory
    if [ -n "$NIPE_DIR" ]; then                                 #check for NIPE, if installed, echo 'NIPE INSTALLED', EXIT.
        echo "[#] NIPE is already installed."
        sleep 1
    else
        echo "NIPE is not installed. Installing now..."

        git clone https://github.com/htrgouvea/nipe             # clone NIPE from git repo
        cd nipe                                                 # cd in NIPE folder
        cpanm --installdeps .                                   # install dependencies
        sudo perl nipe.pl install                               # install NIPE
        echo "[*] NIPE successfully installed."
        sleep 1
        INSTALL
    fi
}
```

- If it does not, it will git clone the installation program, install the dependencies, and start the installation with a sudo admin privileges.
- This is the result if all the required tools or programs are installed.

```
Checking to see if required tools are installed:
[#] NIPE is already installed.
[#] GIT is already installed.
[#] GEOIP-BIN is already installed.
[#] NMAP is already installed.
[#] SSHPASS is already installed.
[#] TOR is already installed.

All required tools good to go!
```

4. Anonymity Check

```
function ANON()                                         # here were create a function ANON
{
    echo
    echo "Standby for anonymity check..."
    echo
    sleep 1
    IP_ADD=$(curl -s ifconfig.io)                       # we curl to get IP address, adding -s to silent the noise
    if [ "$(geoiplookup $IP_ADD | grep -i SG)" ]        # we geoiplookup the IP address, make sure not from SG
    then
        echo "[*] You are NOT anonymous!"               #if SG, we flag as not anonymous
        sleep 2
        NIPE
    else
        echo "[*] You are anonymous!"                   #else, good to go.
        sleep 1
        echo "[*] Spoofed Country: $(geoiplookup $IP_ADD | awk -F: '{print $2}')"       # display spoofed country
        sleep 1
        echo "[*] Spoofed IP: $IP_ADD"                                                  # display spoofed IP
        echo
        sleep 2
        figlet "Good to Go!"
    fi
}
```

- Check if the network connection is anonymous; if not, alert the user and exit.
- If the network connection is anonymous, display the spoofed country name.
- Here we use conditional statement to check if the local machine IP address on line 154 has the word 'SG'.
- If the word 'SG' is present, the program will flag that the user is not anonymous.
- It will then continue to run the NIPE funtion to start the program.

```
172  #~ Create Nipe function here to turn on Nipe.
173  function NIPE()
174  {
175
176      echo "Launching NIPE..."
177      sleep 1
178      echo "Checking Launch Status, Standby..."
179      echo
180      cd nipe                                          # cd into the nipe folder
181      sudo perl nipe.pl status                         # run this command to check the status of nipe
182      sleep 1
183      echo "Launch Status False. Starting NIPE."
184      sudo perl nipe.pl start                          # start the nipe program
185      sudo perl nipe.pl status                         # run this command to check the status of nipe
186      #check if status is false. while status is false, run command to connect.Need function here.
187      sleep 1
188      echo "NIPE is Running.Status True.."
189      sleep 1
190      ANON                                             # run the ANON function back
191  }
```

```
Standby for anonymity check ...

[*] You are NOT anonymous!
Launching NIPE ...
Checking Launch Status, Standby ...


[+] Status: false
[+] Ip: 103.6.150.47

Launch Status False. Starting NIPE.

[+] Status: true
[+] Ip: 185.220.101.27

NIPE is Running.Status True ..
```

- Once Nipe is running, the anonymity check will begin again.
- It will then display the spoofed country and IP address. Here is an example:

```
Standby for anonymity check ...

[*] You are anonymous!
[*] Spoofed Country:  DE, Germany
[*] Spoofed IP: 185.220.101.27
```

5. **Allow the user to specify the address/URL**

```
193  #~ v) Allow the user to specify the address/URL to whois from remote server; save into a variable
194  function TARGET()
195  {
196      read -p "[?] Specify a Domain/IP Address to Attack: " target
197  }
```

- We will then await user's input for address or URL and save it to the 'target' variable
- Here we can see the user's input is 'scanme.nmap.com'

```
[?] Specify a Domain/IP Address to Attack: scanme.nmap.com
```

6. **Scanning the remote machine for open ports**

```
200  #~ 2. Automatically Scan the Remote Server for open ports.
201  function SCAN_REMOTE()
202  {
203      #how to scan? using nmap -Pn?
204      echo "-------------------------------------------"
205      echo
206      echo "[*] Scanning Remote Server for Open Ports.."
207      sleep 2
208      nmap 192.168.254.129                    # scan the remote machine for open ports
209      echo
210      sleep 1
211      echo "[*] Scanning Completed."
212      sleep 1
213
214  }
```

- Using Nmap, we scan the remote machine to ensure that the 2 ports we want to use are open: SSH and FTP.

```
[*] Scanning Remote Server for Open Ports..
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-30 05:16 EDT
Nmap scan report for 192.168.254.129
Host is up (0.0019s latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT   STATE SERVICE
21/tcp open  ftp
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.10 seconds

[*] Scanning Completed.
```

7. Connect to the Remote Server via SSH

```
228  function CONNECT_REMOTE()
229  {
230
231      export SSHPASS='[redacted]' # password to for automatic connection to remote machine, how to store this better?
232      echo
233      echo "[*] Establishing connection to Remote Server..."
234      echo
235      sleep 1
236      RMT_UPTIME          #get the uptime
237      sleep 1
238      RMT_IP              #get the IP Address
239      sleep 1
240      RMT_COUNTRY         #get the Country
241      echo
242      echo "-------------------------------------------"
243      sleep 1
244      echo
245  }
```

- Using SSHPASS, we store the password for automatic SSH connection later.
- We run the functions RMT_UPTIME, RMT_COUNTRY and RMT_IP to display them.

```
[*] Establishing connection to Remote Server...

Uptime:  09:16:49 up 1 day, 23:58,  1 user,  load average: 0.22, 0.15, 0.16
IP Address: [redacted]
Country: [redacted]
```

8. WHOIS and scanning user input via remote server

```
266  # Commence Attack
267  function ATTACK()
268  {
269      sudo touch /var/log/nr.log      #standby and create nr.log
270      echo "Commencing attack from Remote Server in 3.."
271      sleep 1
272      echo "2.."
273      sleep 1
274      echo "1.."
275      sleep 1
276      echo
277      WHOIS_TARGET                    # WHOIS the given URL by user
278      NMAP_TARGET                     # NMAP the given URL by user
279      sleep 1
280      echo
281      echo " ------------------------------------------------------------------------------------------"
282      echo "|                                                                                          |"
283      echo "| Attack Successful. Go to /var/log/nr.log to view log file records of Domain/IP Add scanned |"
284      echo "|                                                                                          |"
285      echo " ------------------------------------------------------------------------------------------"
286      sleep 3
287  }
```

- Function ATTACK is executed automatically after establishing connection to Remote Server
- Firstly, we create a nr.log file in the /var/log directory. This is so we can store data logs of who we WHOIS and scanned.

```
291  function WHOIS_TARGET() # FROM REMOTE SERVER, NOT LOCAL
292  {
293      # how? we whois straight IP_OR_DOMAIN
294      echo "[*] Whoising target's address: $target"
295      # save the data somewhere
296      sshpass -e ssh -q tc@192.168.254.129 "whois '$target' > whois_output.txt"   #~ i) Save the Whois data into file on the remote computer
297      sleep 1
298      wget -q ftp://tc:tc@192.168.254.129/whois_output.txt                       # Download the generated whois_output.txt from remote machine
299      echo "[@] WHOIS data of $target was saved into $(pwd)"                     # We tell Whois data was saved into directory path
300      sudo echo "$(date)- [*] WHOIS data collected for: $target" >> nr.log       # Add log report to nr.log to state what is happening.
301      echo
302      sleep 5
303  }
```

- Here we tell the user that we are going to commence WHOIS on the given URL.
- With the SSHPASS stored earlier, and the username and IP address, we remote into the server to run the whois command and save it onto a text file.
- Next, we wget using ftp into the remote machine to extract the recently saved log file of the whois data.
- We append the log data on nr.log in the same project directory.
- Appending directly to /var/log/nr.log is denied due to permissions.
- The same process is done for Nmap.

```
Commencing attack from Remote Server in 3..
2..
1..

[*] Whoising target's address: scanme.nmap.com
[@] WHOIS data of scanme.nmap.com was saved into /home/kali/Desktop/Project/Project_3/nipe

[*] Scanning target's address: scanme.nmap.com
[@] NMAP data of scanme.nmap.com was saved into /home/kali/Desktop/Project/Project_3/nipe

 ____________________________________________________________________________
|                                                                            |
| Attack Successful. Go to /var/log/nr.log to view log file records of Domain/IP Add scanned |
|                                                                            |
 ____________________________________________________________________________
```

9. Results (additional)

```
321  function NR_LOG()   # move logs from current file to /var/log/nr.log  # so work around, we save the nr.log on project folder, then move it to /var/log
322  {
323      sudo cp nr.log /var/log
324      echo
325      echo "Verifying Accuracy of Data Log in /var/log/nr.log..."
326      echo
327      sleep 1
328      cat /var/log/nr.log
329  }
```

- As mentioned, we were unable to append directly to the /var/log/nr.log.
- Therefore, we copy the nr.log file that is stored in the project folder instead.
- We would then verify the accuracy of the logged data in /var/log/nr.log by using the cat command.

```
Verifying Accuracy of Data Log in /var/log/nr.log ...

Sun May 26 02:54:37 AM EDT 2024- [*] WHOIS data collected for: scanme.nmap.com
Sun May 26 02:55:04 AM EDT 2024- [*] NMAP data collected for: scanme.nmap.com
Sun May 26 10:24:32 AM EDT 2024- [*] WHOIS data collected for: scanme.nmap.com
Sun May 26 10:25:06 AM EDT 2024- [*] NMAP data collected for: scanme.nmap.com
Thu May 30 05:16:58 AM EDT 2024- [*] WHOIS data collected for: scanme.nmap.com
Thu May 30 05:17:26 AM EDT 2024- [*] NMAP data collected for: scanme.nmap.com
```

METHODOLOGIES

SCOPE 2: NETWORK RESEARCH AND MONITORING

1. Capture Network Traffic

Start wireshark to capture packet data:

2. Secure Network Protocol

Installation of SFTP on local machine:

SFTP is part of the OpenSSH package, so the first step is to install OpenSSH. Install Open SSH Server:

sudo apt-get install openssh-server

SSH is also needed to access SFTP server. If it is not installed, we install it with the following command:

sudo apt-get install ssh

Create a Group and User for SFTP: For security reasons, it's a good practice to create a specific group and user for SFTP service:

sudo groupadd sftpg

sudo useradd -g sftpg localMac

sudo passwd localMac

Breakdown
sudo: This is a command that allows you to run programs with the security privileges
groupadd: This is a command that lets you create a new group.
sftpg: This is the name of the group you're creating.
useradd: This is a command that lets you create a new user.
-g sftpg: This option specifies the group that the new user will be added to. In this case, the group is "sftpg".
localMac: This is the name of the user you're creating.
passwd: This is a command that lets you change the password of a user.
localMac: This is the name of the user whose password you're changing.

Installation of SFTP on remote machine:

Install SSH on remote machine if it is not installed:

sudo apt-get install ssh

Create a new user group for SFTP:

sudo addgroup sftp

Create and configure a new user:

sudo useradd -m remoteMac -g sftp

sudo passwd remoteMac

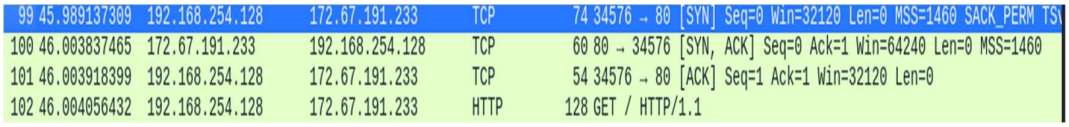
Breakdown
sudo: This is a command that allows you to run programs with the security privileges
useradd: This is a command that lets you create a new user.
sftpg: This is the name of the group you're creating.
useradd: This is a command that lets you create a new user.
-m: This option/flag creates a home directory for the new user.
-g sftp: This option/flag specifies the group that the new user will be added to. In this case, the group is "sftp".
remoteMac: This is the name of the user you're creating in the remote machine.
passwd: This is a command that lets you change the password of a user.
remoteMac: This is the name of the user whose password you're changing.

DISCUSSION

SCOPE 2: NETWORK RESEARCH AND MONITORING

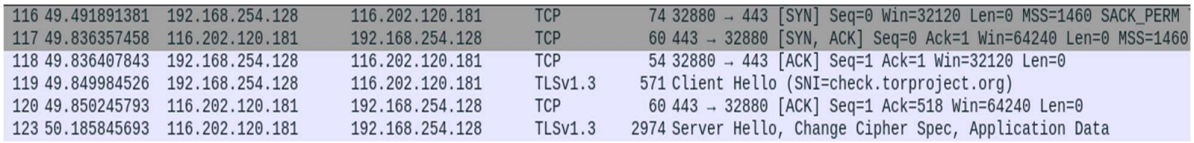
1. Capture Network Traffic

3-way handshake of getting curl from ifconfig.io:



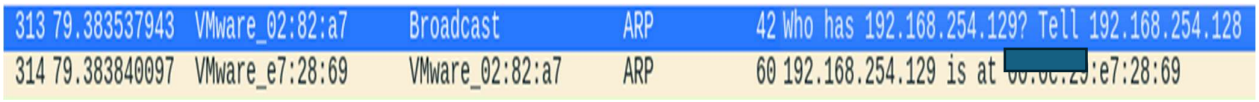
99	45.989137309	192.168.254.128	172.67.191.233	TCP	74 34576 → 80 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM TS
100	46.003837465	172.67.191.233	192.168.254.128	TCP	60 80 → 34576 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
101	46.003918399	192.168.254.128	172.67.191.233	TCP	54 34576 → 80 [ACK] Seq=1 Ack=1 Win=32120 Len=0
102	46.004056432	192.168.254.128	172.67.191.233	HTTP	128 GET / HTTP/1.1

Establishing connection to Nipe



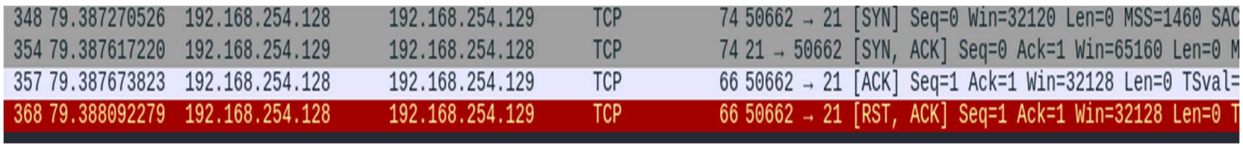
116	49.491891381	192.168.254.128	116.202.120.181	TCP	74 32880 → 443 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM
117	49.836357458	116.202.120.181	192.168.254.128	TCP	60 443 → 32880 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
118	49.836407843	192.168.254.128	116.202.120.181	TCP	54 32880 → 443 [ACK] Seq=1 Ack=1 Win=32120 Len=0
119	49.849984526	192.168.254.128	116.202.120.181	TLSv1.3	571 Client Hello (SNI=check.torproject.org)
120	49.850245793	116.202.120.181	192.168.254.128	TCP	60 443 → 32880 [ACK] Seq=1 Ack=518 Win=64240 Len=0
123	50.185845693	116.202.120.181	192.168.254.128	TLSv1.3	2974 Server Hello, Change Cipher Spec, Application Data

ARP with remote machine:

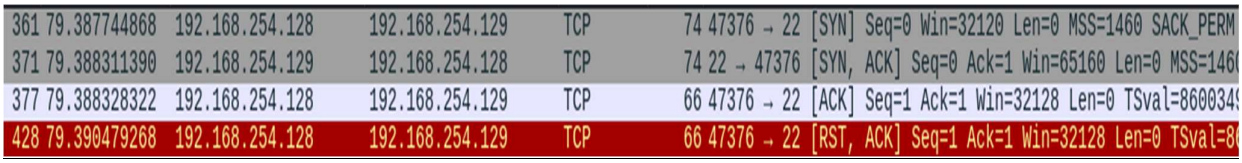


313	79.383537943	VMware_02:82:a7	Broadcast	ARP	42 Who has 192.168.254.129? Tell 192.168.254.128
314	79.383840097	VMware_e7:28:69	VMware_02:82:a7	ARP	60 192.168.254.129 is at [redacted]:e7:28:69

Scanning of open ports for remote machine (Port 21 & 22):

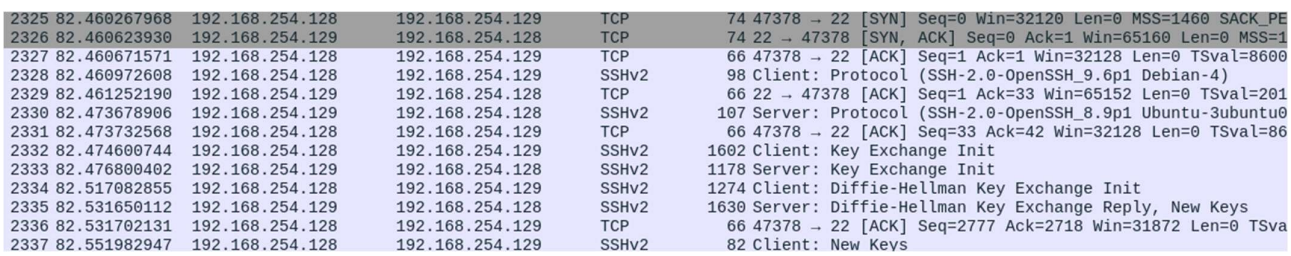


348	79.387270526	192.168.254.128	192.168.254.129	TCP	74 50662 → 21 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SAC
354	79.387617220	192.168.254.129	192.168.254.128	TCP	74 21 → 50662 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 M
357	79.387673823	192.168.254.128	192.168.254.129	TCP	66 50662 → 21 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=
368	79.388092279	192.168.254.128	192.168.254.129	TCP	66 50662 → 21 [RST, ACK] Seq=1 Ack=1 Win=32128 Len=0 T



361	79.387744868	192.168.254.128	192.168.254.129	TCP	74 47376 → 22 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PERM
371	79.388311390	192.168.254.129	192.168.254.128	TCP	74 22 → 47376 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=146
377	79.388328322	192.168.254.128	192.168.254.129	TCP	66 47376 → 22 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=860034
428	79.390479268	192.168.254.128	192.168.254.129	TCP	66 47376 → 22 [RST, ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=8

First SSH connection to remote machine via SSHPASS:



2325	82.460267968	192.168.254.128	192.168.254.129	TCP	74 47378 → 22 [SYN] Seq=0 Win=32120 Len=0 MSS=1460 SACK_PE
2326	82.460623930	192.168.254.129	192.168.254.128	TCP	74 22 → 47378 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1
2327	82.460671571	192.168.254.128	192.168.254.129	TCP	66 47378 → 22 [ACK] Seq=1 Ack=1 Win=32128 Len=0 TSval=8600
2328	82.460972608	192.168.254.128	192.168.254.129	SSHv2	98 Client: Protocol (SSH-2.0-OpenSSH_9.6p1 Debian-4)
2329	82.461252190	192.168.254.129	192.168.254.128	TCP	66 22 → 47378 [ACK] Seq=1 Ack=33 Win=65152 Len=0 TSval=201
2330	82.473678906	192.168.254.129	192.168.254.128	SSHv2	107 Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0
2331	82.473732568	192.168.254.128	192.168.254.129	TCP	66 47378 → 22 [ACK] Seq=33 Ack=42 Win=32128 Len=0 TSval=86
2332	82.474600744	192.168.254.128	192.168.254.129	SSHv2	1602 Client: Key Exchange Init
2333	82.476800402	192.168.254.129	192.168.254.128	SSHv2	1178 Server: Key Exchange Init
2334	82.517082855	192.168.254.128	192.168.254.129	SSHv2	1274 Client: Diffie-Hellman Key Exchange Init
2335	82.531650112	192.168.254.129	192.168.254.128	SSHv2	1630 Server: Diffie-Hellman Key Exchange Reply, New Keys
2336	82.531702131	192.168.254.128	192.168.254.129	TCP	66 47378 → 22 [ACK] Seq=2777 Ack=2718 Win=31872 Len=0 TSva
2337	82.551982947	192.168.254.128	192.168.254.129	SSHv2	82 Client: New Keys

Connection to remote machine via ftp:

```
2527 91.651897816  192.168.254.128  192.168.254.129  FTP   75 Request: USER tc
2528 91.652078650  192.168.254.129  192.168.254.128  TCP   66 21 → 50964 [ACK] Seq=21 Ack=10 Win=65280 Len=0 TSva
2529 91.652205281  192.168.254.129  192.168.254.128  FTP  100 Response: 331 Please specify the password.
2530 91.652278230  192.168.254.128  192.168.254.129  FTP   75 Request: PASS tc
2531 91.676888825  192.168.254.129  192.168.254.128  FTP   89 Response: 230 Login successful.
2532 91.677022114  192.168.254.128  192.168.254.129  FTP   72 Request: SYST
2533 91.677425285  192.168.254.129  192.168.254.128  FTP   85 Response: 215 UNIX Type: L8
2534 91.677511033  192.168.254.128  192.168.254.129  FTP   71 Request: PWD
2535 91.677852477  192.168.254.129  192.168.254.128  FTP  107 Response: 257 "/home/tc" is the current directory
2536 91.677954496  192.168.254.128  192.168.254.129  FTP   74 Request: TYPE I
2537 91.678421620  192.168.254.129  192.168.254.128  FTP   97 Response: 200 Switching to Binary mode.
2538 91.678537906  192.168.254.128  192.168.254.129  FTP   89 Request: SIZE whois_output.txt
2539 91.678919912  192.168.254.129  192.168.254.128  FTP   76 Response: 213 2240
2540 91.679019677  192.168.254.128  192.168.254.129  FTP   72 Request: PASV
```

2. Research on FTP

Purpose & Features:

FTP (File Transfer Protocol) is a standard communication protocol used for transferring files between computers on a TCP/IP network. It's designed to promote sharing of files, encourage indirect use of remote computers, and shield users from system variations. FTP supports bi-directional data transfer, allowing files to be uploaded and downloaded between a client and server. It also supports batch file transfer, enabling multiple files to be transferred in one go.

RFCs (Request for Comments):

The original specification for the File Transfer Protocol was written by Abhay Bhushan and published as RFC 114 on 16 April 1971. The protocol was later replaced by a TCP/IP version, RFC 765 (June 1980) and RFC 959 (October 1985), the current specification. Several proposed standards amend RFC 959, for example RFC 1579 (February 1994) enables Firewall-Friendly FTP (passive mode), RFC 2228 (June 1997) proposes security extensions, RFC 2428 (September 1998) adds support for IPv6 and defines a new type of passive mode.

Behavior & Message Exchange Process:

FTP runs on top of TCP, like HTTP. To transfer a file, 2 TCP connections are used by FTP in parallel: control connection and data connection. The control connection is used for sending control information like user identification, password, commands to change the remote directory, commands to retrieve and store files, etc. The data connection is used for the actual file transfer.

Mechanisms:

The header of a TCP segment, which FTP uses, can range from 20-60 bytes. FTP commands comprise the control information flowing from the user-FTP to the server-FTP process. FTP can transfer ASCII, EBCDIC, or image files. The ASCII is the default file share format, in this, each character is encoded by NVT ASCII. The image file format is the default format for transforming binary files.

Strengths & Weaknesses (CIA Triad):

The strengths of FTP include its simplicity, speed, and robustness, making it an excellent choice for basic file transfer needs. It also shields the user from system variations (operating system, directory structures, file structures, etc.). However, FTP's weaknesses include lack of encryption in its basic form, making it vulnerable to data interception and unauthorized access. In terms of the CIA Triad, FTP can ensure confidentiality and integrity when secured with SSL/TLS (FTPS) or replaced with SSH File Transfer Protocol (SFTP). This will be discussed further later. Availability is ensured as long as the FTP server is running and accessible.

3. Secure Network Protocol

SFTP:

As mentioned earlier, we will be taking a look at SSH File Transfer Protocol (SFTP). SFTP offers all the functionality of FTP, but it does it over a secure channel. This means that no passwords or file data are transferred in clear text. All communications are encrypted, ensuring data confidentiality and integrity.

Even thought there are more steps are that are being taken, it ensures data confidentiality and integrity inline with the CIA triad.

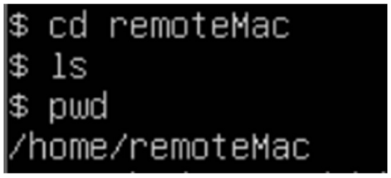
4. Execution of SFTP usage

Start wireshark to capture packet data:

Create text file on remote machine to test the connection:

First, we change the user to remoteMac, insert the password

Next, we cd to the /home/remoteMac directory.



We then create a file “downloadthis.txt”.

```
$ touch downloadthis.txt
$ ls
downloadthis.txt
```

Connecting to remote server from local machine and downloading:

Go to the location in the local machine we want to download the items into from the remote server. In this instance, it’s the download folder.

```
┌──(kali㉿kali)-[~]
└─$ cd Downloads

┌──(kali㉿kali)-[~/Downloads]
└─$ ls
23-Monitor.pcap   auth.log       project3.pcapng  'Transcript Let the cash flow.txt'
auth2.log         auth_old.log   robots.txt       'Transkrip Let the Cash Flow.txt'
auth2.log.1       linux_2k.log   text              wordlist.txt
```

Connect to the remote server via SFTP in the following manner:
sftp <username of remote server>@<IP address of remote server>

Insert the password of the remote server username.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sftp remoteMac@192.168.254.129
remoteMac@192.168.254.129's password:
Connected to 192.168.254.129.
```

List the items that is available on the /home/remoteMac directory.
Download the ‘downloadthis.txt’ file using:
get downloadthis.txt

```
sftp> ls
downloadthis.txt
sftp> get downloadthis.txt
Fetching /home/remoteMac/downloadthis.txt to downloadthis.txt
```

Exit and check if item is downloaded.

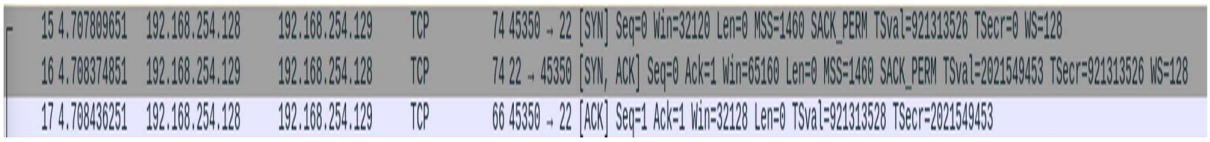
```
sftp> exit

┌──(kali㉿kali)-[~/Downloads]
└─$ ls
23-Monitor.pcap   auth.log           linux_2k.log      text                                wordlist.txt
auth2.log         auth_old.log       project3.pcapng  'Transcript Let the cash flow.txt'
auth2.log.1       downloadthis.txt   robots.txt       'Transkrip Let the Cash Flow.txt'
```

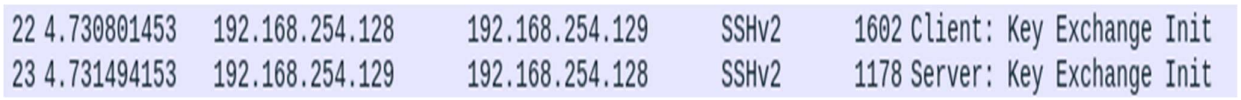
5. **Analyse pcap file captured by wireshark**

Usage of ARP to determine remote server's address:

Establishing connection between remote and local machine using 3-way handshake:

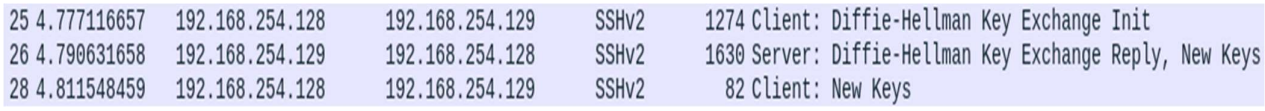


Client-server key exchange initialisation:



The key exchange initiation is a crucial part of establishing a secure SSH connection. It involves both the client and the server generating temporary public-private key pairs and sharing their respective public keys to produce a shared secret key. This shared secret key is then used to encrypt the rest of the communication during the SSH session.

Client-server Diffie-Hellman Key Exchange Init:



"Client: Diffie-Hellman Key Exchange Init" indicates that the client has initiated the key exchange process by generating a private key and corresponding public key and sending the public key to the server. "Server: Diffie-Hellman Key Exchange Reply, New Keys" indicates that the server has received the client's public key, used it with its own private key to generate the shared secret key, and is ready to begin secure communication.

When the client sends the "New Keys" message to the server, it is essentially signaling to the server that it has finished computing the shared secret key and is ready to start using it for encrypting further communication. Client is saying: "Hey server! All the following messages from me will use the ciphers we just negotiated."

6. Execution of FTP usage

Start wireshark to capture packet data:

```
┌──(kali㉿kali)-[~/Downloads]
└─$ wireshark
```

Create text file on remote machine to test the connection:

We first create a file “ftp.txt”.

```
tc@server:~$ touch ftp.txt
tc@server:~$ ls
auth.log  ftp.txt  Later  LiNuX
```

Connecting to remote server from local machine and downloading the file:

Go to the location in the local machine we want to download the items into from the remote server. In this instance, it’s the download folder.

```
┌──(kali㉿kali)-[~]
└─$ cd Downloads

┌──(kali㉿kali)-[~/Downloads]
└─$ ls
23-Monitor.pcap   auth.log       project3.pcapng  'Transcript Let the cash flow.txt'
auth2.log         auth_old.log   robots.txt       'Transkrip Let the Cash Flow.txt'
auth2.log.1       linux_2k.log   text              wordlist.txt
```

Connect to the remote server via SFTP in the following manner, followed by the password:
ftp <username of remote server>@<IP address of remote server>

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ftp tc@192.168.254.129
Connected to 192.168.254.129.
220 (vsFTPd 3.0.5)
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
```

Using **get ftp.txt**, we download the ftp.txt file to our local machine and exit.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ls
23-Monitor.pcap     project3.pcapng
auth2.log           project3_sftp.pcapng
auth2.log.1         robots.txt
auth.log            text
auth_old.log       'Transcript Let the cash flow.txt'
downloadthis.txt   'Transkrip Let the Cash Flow.txt'
ftp.txt             wordlist.txt
linux_2k.log
```

7. Analyse pcap file captured by wireshark

Establishing connection between remote and local machine using 3-way handshake:

10	5.786127932	192.168.254.128	192.168.254.129	TCP	74 39076 → 21 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM TSval=929274442 TSecr=0 WS=2
11	5.786448629	192.168.254.129	192.168.254.128	TCP	74 21 → 39076 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2022064434 TSecr=929274442 WS=128
12	5.786482929	192.168.254.128	192.168.254.129	TCP	66 39076 → 21 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=929274442 TSecr=2022064434

As all TCP connection, a 3-way handshake is initially established.

Establishing connection between remote and local machine using 3-way handshake:

13	5.791404494	192.168.254.129	192.168.254.128	FTP	86 Response: 220 (vsFTPd 3.0.5)
14	5.791456493	192.168.254.128	192.168.254.129	TCP	66 39076 → 21 [ACK] Seq=1 Ack=21 Win=65516 Len=0 TSval=929274447 TSecr=2022064439
15	5.791765791	192.168.254.128	192.168.254.129	FTP	75 Request: USER tc
16	5.792053389	192.168.254.129	192.168.254.128	TCP	66 21 → 39076 [ACK] Seq=21 Ack=10 Win=65280 Len=0 TSval=2022064439 TSecr=929274448
17	5.792199788	192.168.254.129	192.168.254.128	FTP	100 Response: 331 Please specify the password.
18	5.838846752	192.168.254.128	192.168.254.129	TCP	66 39076 → 21 [ACK] Seq=10 Ack=55 Win=65482 Len=0 TSval=929274495 TSecr=2022064439
21	7.780972280	192.168.254.128	192.168.254.129	FTP	75 Request: PASS tc
22	7.795941176	192.168.254.129	192.168.254.128	FTP	89 Response: 230 Login successful.

Remote machine response with a successful FTP connection. Here we can see local machine connecting to remote machine with the username and password being displayed, before a successful login.

Establishing connection between remote and local machine:

13	5.791404494	192.168.254.129	192.168.254.128	FTP	86 Response: 220 (vsFTPd 3.0.5)
14	5.791456493	192.168.254.128	192.168.254.129	TCP	66 39076 → 21 [ACK] Seq=1 Ack=21 Win=65516 Len=0 TSval=929274447 TSecr=2022064439
15	5.791765791	192.168.254.128	192.168.254.129	FTP	75 Request: USER tc
16	5.792053389	192.168.254.129	192.168.254.128	TCP	66 21 → 39076 [ACK] Seq=21 Ack=10 Win=65280 Len=0 TSval=2022064439 TSecr=929274448
17	5.792199788	192.168.254.129	192.168.254.128	FTP	100 Response: 331 Please specify the password.
18	5.838846752	192.168.254.128	192.168.254.129	TCP	66 39076 → 21 [ACK] Seq=10 Ack=55 Win=65482 Len=0 TSval=929274495 TSecr=2022064439
21	7.780972280	192.168.254.128	192.168.254.129	FTP	75 Request: PASS tc
22	7.795941176	192.168.254.129	192.168.254.128	FTP	89 Response: 230 Login successful.

Remote machine response with a successful FTP connection. Here we can see local machine connecting to remote machine with the username and password being displayed, before a successful login.

File request:

```
80 Request: MDTM ftp.txt
```

The files that were requested and downloaded can also be seen in the analysis.

CONCLUSION

In this Network Research project, we delved into two main areas: Creating an automation that would allow us to run the script from our local device but have it executed by a remote server anonymously, and capturing the traffic during the automated attack on the server for further analysis.

1. **Scope 1: Network Remote control**

By creating an automation that allows us to run scripts from our local device but have them executed by a remote server anonymously, we aim to understand how this whole process works:

Control and Customization: Understanding the workings of the automation allows for better control and customization of the process to suit specific needs and scenarios.

Troubleshooting and Optimization: If issues arise or improvements are needed, understanding the underlying process is essential for effective troubleshooting and optimization.

Security Enhancements: A deep understanding of the process can reveal potential security vulnerabilities that might be exploited by malicious actors. These can then be addressed to further enhance the security of the system.

Knowledge Transfer: Understanding the process enables the practitioner to effectively communicate and explain the system to other team members, stakeholders, or during incident response.

Simulating Attacker Techniques: This project helps us stay ahead of the curve by gaining understanding of how attackers operate. This knowledge can help us anticipate the attackers tactics and develop more effective contermeasures.

Investigating Emerging Threats: As new attack methods emerge, we can use tthe automation to test our defenses against them. This allows us to proactively identify areas where our security posture might be inadequate and take steps to address them before attackers can leverage those vulnerabilities.

In conclusion, both the implementation of this project and a thorough understanding of its workings are vital in enhancing network security operations, improving response times to threats, and ultimately safeguarding the integrity of our digital assets. This project underscores the importance of continuous learning, adaptation, and innovation in the ever-evolving landscape of cybersecurity.

2. **Scope 2: Network Research and Monitoring**

When it comes to choosing between FTP and SFTP for file transfer, security is the paramount concern. FTP transmits data in plain text, leaving usernames, passwords, and the transferred information vulnerable to interception. This makes it unsuitable for any sensitive data exchange.

On the other hand, SFTP leverages SSH encryption, creating a secure tunnel for data transfer. Usernames, passwords, and the transferred files are all encrypted, rendering them unreadable by anyone snooping on the network traffic. This robust security mechanism makes SFTP the clear winner for safeguarding sensitive information during file transfers.

While FTP offers a simpler setup, its glaring security shortcomings outweigh this advantage. For secure file transfers, SFTP is the uncontested choice. It ensures the confidentiality and integrity of your data, protecting both usernames/passwords and the transferred items from unauthorized access. In today's security-conscious environment, SFTP stands as the reliable and secure method for file transfer.

RECOMMENDATIONS

Security Enhancements:

SFTP over FTP: While the script utilizes SSH for the initial connection, it retrieves data via FTP. This is a significant security concern. Strongly recommend switching to SFTP (SSH File Transfer Protocol). SFTP encrypts both data and commands during transfer, offering a more secure alternative to FTP.

Authorization and Permissions: The script mentions "proper authorization" for attacks. Ensure the script uses proper user accounts and adheres to access controls on the remote server. Avoid hardcoded credentials and implement secure authentication methods. (I.E here SSHPASS is hardcoded)

Data Security: The script saves collected data (Whois & Nmap) on the remote server before retrieval. Consider the security implications of storing this data. Explore options like temporary storage or direct transfer to the local machine.

Logging and Auditing: While the script creates a log for auditing, ensure the logs themselves are stored securely and don't contain sensitive information.

REFERENCES

1. OpenAI. ChatGPT. Personal communication. May 27, 2024.

2. Gemini (Google AI, 2024). Personal communication, May 27, 2024.

3. Microsoft Copilot. Microsoft. Personal communication. May 27, 2024

4. Date Command in Linux: How to Set, Change, Format and Display Date
   https://phoenixnap.com/kb/linux-date-command

5. Wget Command in Linux with Examples
   https://linuxize.com/post/wget-command-examples/

6. Start Learning Linux
   https://linuxhandbook.com/

7. How to search and extract string from command output?
   https://unix.stackexchange.com/questions/393948/how-to-search-and-extract-string-from-command-output

8. SSH password automation in Linux with sshpass
   https://www.redhat.com/sysadmin/ssh-automation-sshpass

9. A Guide to RFCs
   https://bluegoatcyber.com/blog/a-guide-to-rfcs/

10. File Transfer Protocol (FTP)
    https://www.geeksforgeeks.org/file-transfer-protocol-ftp/

11. The Ultimate Guide to FTP, FTPS and SFTP
    https://contabo.com/blog/the-ultimate-guide-to-ftp-ftps-and-sftp/

12. How To: Use WIRESHARK to find out which files are downloaded from a server through HTTP/UNC
    https://forums.ivanti.com/s/article/How-To-Use-WIRESHARK-to-find-out-which-files-are-downloaded-from-a-server-through-HTTP-UNC?language=en_US

13. Understand wireshark capture for ssh key exchange
    https://serverfault.com/questions/586638/understand-wireshark-capture-for-ssh-key-exchange