Remote Control

Network Research

Edwin Lum
S8
CFC130124

TABLE OF CONTENTS

Introduction

In an era dominated by digital information and online presence, cybersecurity measures are essential in safeguarding sensitive data and ensuring the integrity of networks and systems. With the increasing reliance on remote servers for operations, it becomes imperative to develop tools and scripts that facilitate secure and anonymous execution of tasks, while ensuring efficiency and reducing errors in executing repetitive tasks.

The primary objective of this project is to develop a Bash script to automate the login to a remote server, and execute tasks anonymously, which in this case, is the scanning of a domain of interest.

This report introduces the functionality and purpose of the Remote Control Bash script, automating the scanning of a domain on a remote server. The script is crafted with the goal of enhancing privacy, minimising the risk of exposure, and streamlining the execution of essential operations.

This script will help cybersecurity professionals to streamline their workflow, and improve security with the anonymous execution of tasks on remote servers. The script also serves as a valuable tool to enhance cybersecurity operations, through proactive threat monitoring of domains.

Methodologies

Bash Script

This Remote Control Bash script was written to help users automate the scanning of a domain anonymously on a remote server with the following steps in the script:

1) Installation of required applications (if needed)
2) Connecting machine to Tor network via nipe
3) Get user input for domain/IP address to scan
4) Connect to remote server and display details of remote server
5) Scan the domain and save information into a file before transferring to the local machine
6) Create a log file on the scanned domain

```
#!/bin/bash

#Part 1: This portion of the code checks if the required applications are installed.
#If the applications are not found in the system, they will be installed.
#Otherwise, a message will indicate that the application has already been installed.

sudo apt-get -qq update

if [ "$(apt-cache policy geoip-bin | grep Installed | sed 's/^.*: //')" == '(none)' ]
then
    sudo apt-get install -y geoip-bin
else
    echo '[#] geoip-bin is already installed.'
fi

if [ "$(apt-cache policy tor | grep Installed | sed 's/^.*: //')" == '(none)' ]
then
    sudo apt-get install -y tor
else
    echo '[#] tor is already installed.'
fi

if [ "$(apt-cache policy sshpass | grep Installed | sed 's/^.*: //')" == '(none)' ]
then
    sudo apt-get install -y sshpass
else
    echo '[#] sshpass is already installed.'
fi

if [ "$(ls | grep nipe)" == 'nipe' ]
then
    echo '[#] nipe is already installed.'
else
    git clone https://github.com/htrgouvea/nipe && cd nipe
    sudo apt-get install -y cpanminus
    cpanm --installdeps .
    sudo cpanm install Switch JSON LWP::UserAgent Config::Simple
    sudo perl nipe.pl install
fi
```

Figure 1: Part 1 of Remote Control Bash script

The echo command was widely used in the script to inform the user which part of the script is being run and the outcome after running the commands.

There are a few echo commands used between certain parts of the script to print an extra empty line, allowing the script to be visually clearer when run.

Part 1 of the script first uses the command, sudo apt-get -qq update, to refresh the local package index with the most recent packages from the repositories in preparation for the installation of any required applications. The -qq flag sets the quiet level and reduces most of the output from the command.

The script then uses if else statements to determine whether the applications required to run the script are installed. The condition in the statement used the command, apt-cache policy <application> | grep Installed | sed 's/^.*: //', which uses regex to filter for the string '(none)' to check if the required applications were installed.

If the condition is true, the installation of the application proceeds using the command, sudo apt-get install -y <application>. The -y flag automatically answers yes to all prompts, to allow consecutive installation of applications without any pause.
If the condition is false, the user will be alerted that the application has already been installed.

Nipe cannot be detected using the apt-cache policy command as it is not installed via the package manager on Linux. Hence, the command, ls | grep nipe, was used to detect the presence of nipe directory in the user's default directory to check if nipe was installed.

```
41  #Part 2: This portion of the code connects the machine to the Tor network via nipe.
42  #If connection to nipe is successful and anonymous, the spoofed IP and Country will be displayed.
43  #If connection fails, the script will alert the user and exit.
44
45  cd ~/nipe
46
47  sudo perl nipe.pl restart
48
49  anonip=$(sudo perl nipe.pl status | grep Ip | awk '{print$3}')
50
51  if [ "$(sudo perl nipe.pl status | grep Status | awk '{print$3}')" == "true" ]
52  then
53      echo "[*] You are anonymous..Connecting to the remote Server."
54      echo
55      echo "[*] Your spoofed IP address is: $anonip, Spoofed country: $(geoiplookup $anonip | grep , | sed 's/^.*, //')"
56  else
57      echo '[-] You are unable to connect anonymously, exiting script.'
58      exit
59  fi
```

Figure 2: Part 2 of Remote Control Bash script

Part 2 of the script first changes directory to ~/nipe to allow the next command, sudo perl nipe.pl restart, to connect to the Tor network to spoof the machine's IP address and country. This command allows the user to immediately run the script to run again if the connection was unsuccessful, without going to the nipe directory to use sudo perl nipe.pl stop.

The anonip variable is used to store the spoofed IP address as it will be called out and used multiple times in the script.

The if else statement checks whether the connection to nipe is successful. The command, sudo perl nipe.pl status | grep Status | awk '{print$3}', filters for the string ‘true’ to check if the machine is connected to nipe.

If the condition is true, the user will be informed of the successful connection and the spoofed IP address is displayed using the variable, anonip. The command, geoiplookup $anonip | grep , | sed 's/^.*, //', uses regex to filter for the string after ‘, ’ in the line, to print out the spoofed country as an output.
If the condition is false, the script will be exited automatically.

```
61  #Part 3: Get user input for Domain/IP to scan
62
63  echo -n '[?] Specify a Domain/IP address to scan: ' ; read doip
64  echo
```

Figure 3: Part 3 of Remote Control Bash script

Part 3 of the script prints out a line asking for user input or a domain or IP address to scan, while reading the input. The -n flag for echo ensures that the input of the user is on the same line as the printed line. The input of the user is stored in the variable, doip.

```
66  #Part 4: Connect automatically to remote server, and display details of the remote server (uptime, IP and country)
67  #The Domain/IP given by user is scanned and data is appended into a file 'whois_doip'
68  #The file is then transferred into your local machine using FTP
69  #REPLACE 'tc' in the variables with the username(under 'user' variable) and password(under 'pw' variable) to your remote server!
70
71  user=tc
72  pw=tc
73  remote=192.168.163.129
74
75  sshpass -p $pw ssh -q -t -o StrictHostKeyChecking=no $user@$remote "echo '[*] Connecting to remote server:'
76  echo -n "Uptime:" && uptime
77  echo "IP Address: $remote"
78  echo "Country: $(whois $remote | grep -i country | awk '{print$2}')"
79  echo
80
81  whois $doip >> whois_doip .
82  echo "[*] Extracting data using whois on Domain/IP address of victim"
83  echo "[@] whois data is saved into /home/kali/nipe/whois_doip"
84  exit"
85
86  curl -s -u tc:tc -O ftp://$remote/whois_doip
```

Figure 4: Part 4 of Remote Control Bash script

Part 4 of the script first defines the variables to be used as the username, password and IP address of the remote server for sshpass. It would be easier to adjust the required input as variables rather than embedding them as part of the code.

The command, sshpass, was used to automatically connect to a remote server via secure shell (SSH). The -p flag allowed for the password to be passed as an argument. The -q flag activates quiet mode, removing most messages sent from the server. The -t flag was used to allocate a pseudo-terminal to the command. The -o flag allows the option, StrictHostKeyChecking=no, to bypass the inputs required when logging in to a remote server for the first time.
The commands after sshpass are in double quotation marks to ensure that they are able to run on the remote server.

The -n flag on line 76 was used to allow the command, uptime, to be on the same line as the printed string. The uptime command prints the current time, the length of time the system has been up, the number of users online, and the load average.

The IP address of the remote server was printed using the stored variable, remote, and the country was filtered with the grep command after using the whois command on the same variable. The -i flag filtered for the word, ignoring the case-sensitivity in the output.

The domain stored in the variable, doip, is scanned using the whois command, and the details are appended into the file, whois_doip, saved on the remote server.

The command, curl -s -u tc:tc -O ftp://$remote/whois_doip, is then used to automatically connect to the remote server via File Transfer Protocol (FTP) to transfer the file, whois_doip, onto the local machine. The -s flag activates silent mode, removing most output messages from the command. The -u flag allows for the username and password to be passed as arguments, ensuring the command runs automatically. The -O flag writes the output to a file.

```
88  #Part 5: A log file is prepared, for the scanned whois data. The data is appended into 'nr.log'
89  echo "$(date) $(echo "[*] whois data collected for: $doip")" >> nr.log
```

Figure 5: Part 5 of Remote Control Bash script

Part 5 of the script creates a log file whenever a domain is scanned using this script. The log file contains the date and time of the scan and the IP address of the domain, which are subsequently appended into the log file, nr.log.

```
┌──(kali㉿kali)-[~]
└─$ bash remotecontrol.sh
[#] geoip-bin is already installed.
[#] tor is already installed.
[#] sshpass is already installed.
[#] nipe is already installed.
[*] You are anonymous..Connecting to the remote Server.

[*] Your spoofed IP address is: 192.42.116.177, Spoofed country: Netherlands
[?] Specify a Domain/IP address to scan: johnbryce.co.il

[*] Connecting to remote server:
Uptime: 13:22:43 up 17:15,  2 users,  load average: 0.00, 0.00, 0.00
IP Address: 192.168.163.129
Country: US

[*] Extracting data using whois on Domain/IP address of victim
[@] whois data is saved into /home/kali/nipe/whois_doip
```

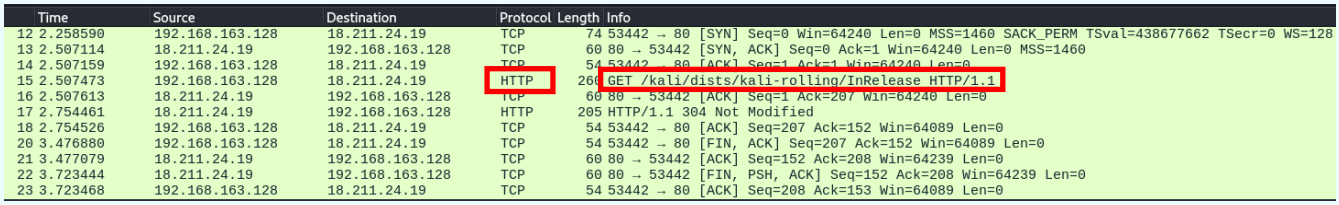
Figure 6: Output for the script when all applications are already installed

```
└─$ cat nr.log
Wed Mar 20 11:33:32 AM EDT 2024 [*] whois data collected for: johnbryce.co.il
```

Figure 7: Output for nr.log

Analysis of network traffic for the Bash script

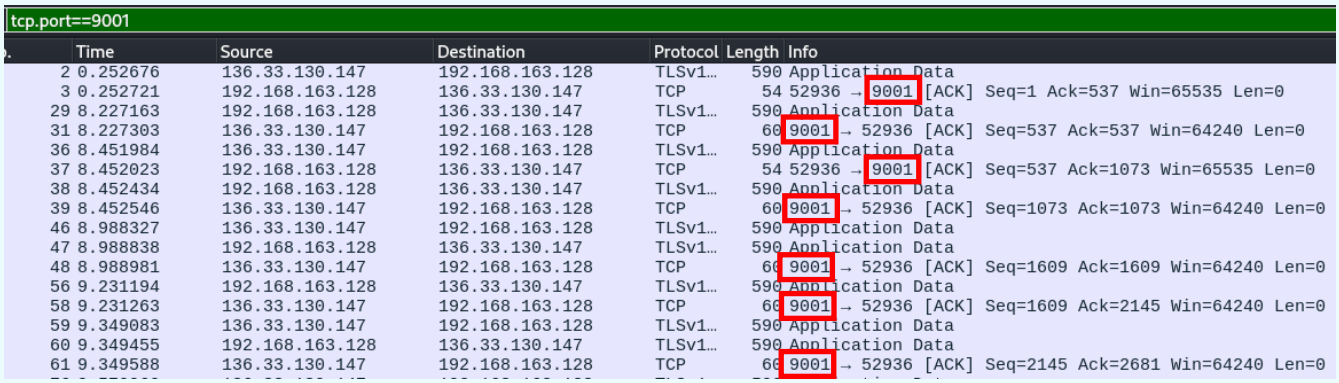
The command, tcpdump -i eth0 -w <filename.pcap>, was used to capture the network packets.



	Time	Source	Destination	Protocol	Length	Info
12	2.258590	192.168.163.128	18.211.24.19	TCP	74	53442 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=438677662 TSecr=0 WS=128
13	2.507114	18.211.24.19	192.168.163.128	TCP	60	80 → 53442 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
14	2.507159	192.168.163.128	18.211.24.19	TCP	54	53442 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
15	2.507473	192.168.163.128	18.211.24.19	HTTP	206	GET /kali/dists/kali-rolling/InRelease HTTP/1.1
16	2.507613	18.211.24.19	192.168.163.128	TCP	60	80 → 53442 [ACK] Seq=1 Ack=207 Win=64240 Len=0
17	2.754461	18.211.24.19	192.168.163.128	HTTP	205	HTTP/1.1 304 Not Modified
18	2.754526	192.168.163.128	18.211.24.19	TCP	54	53442 → 80 [ACK] Seq=207 Ack=152 Win=64089 Len=0
20	3.476880	192.168.163.128	18.211.24.19	TCP	54	53442 → 80 [FIN, ACK] Seq=207 Ack=152 Win=64089 Len=0
21	3.477079	18.211.24.19	192.168.163.128	TCP	60	80 → 53442 [ACK] Seq=152 Ack=208 Win=64239 Len=0
22	3.723444	18.211.24.19	192.168.163.128	TCP	60	80 → 53442 [FIN, PSH, ACK] Seq=152 Ack=208 Win=64239 Len=0
23	3.723468	192.168.163.128	18.211.24.19	TCP	54	53442 → 80 [ACK] Seq=208 Ack=153 Win=64089 Len=0

Figure 8: packet capture for apt-get update

sudo apt-get update connects to the kali website via HTTP to update the package index files on the system.

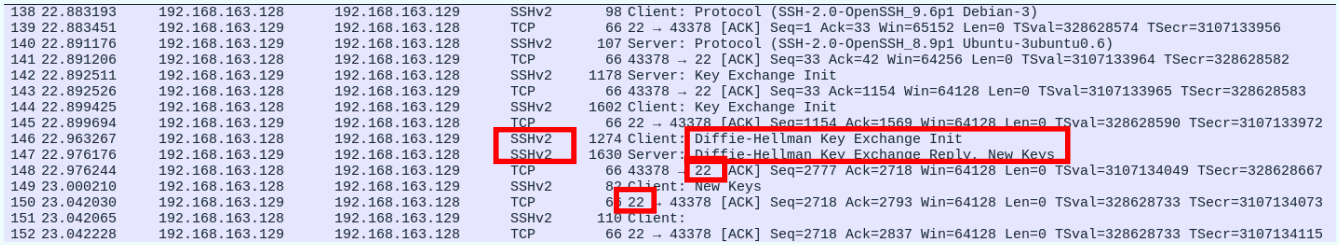


tcp.port==9001

	Time	Source	Destination	Protocol	Length	Info
2	0.252676	136.33.130.147	192.168.163.128	TLSv1…	590	Application Data
3	0.252721	192.168.163.128	136.33.130.147	TCP	54	52936 → 9001 [ACK] Seq=1 Ack=537 Win=65535 Len=0
29	8.227163	192.168.163.128	136.33.130.147	TLSv1…	590	Application Data
31	8.227303	136.33.130.147	192.168.163.128	TCP	60	9001 → 52936 [ACK] Seq=537 Ack=537 Win=64240 Len=0
36	8.451984	136.33.130.147	192.168.163.128	TLSv1…	590	Application Data
37	8.452023	192.168.163.128	136.33.130.147	TCP	54	52936 → 9001 [ACK] Seq=537 Ack=1073 Win=65535 Len=0
38	8.452434	192.168.163.128	136.33.130.147	TLSv1…	590	Application Data
39	8.452546	136.33.130.147	192.168.163.128	TCP	60	9001 → 52936 [ACK] Seq=1073 Ack=1073 Win=64240 Len=0
46	8.988327	136.33.130.147	192.168.163.128	TLSv1…	590	Application Data
47	8.988838	192.168.163.128	136.33.130.147	TLSv1…	590	Application Data
48	8.988981	136.33.130.147	192.168.163.128	TCP	60	9001 → 52936 [ACK] Seq=1609 Ack=1609 Win=64240 Len=0
56	9.231194	192.168.163.128	136.33.130.147	TLSv1…	590	Application Data
58	9.231263	136.33.130.147	192.168.163.128	TCP	60	9001 → 52936 [ACK] Seq=1609 Ack=2145 Win=64240 Len=0
59	9.349083	136.33.130.147	192.168.163.128	TLSv1…	590	Application Data
60	9.349455	192.168.163.128	136.33.130.147	TLSv1…	590	Application Data
61	9.349588	136.33.130.147	192.168.163.128	TCP	60	9001 → 52936 [ACK] Seq=2145 Ack=2681 Win=64240 Len=0

Figure 9: packet capture for Port 9001

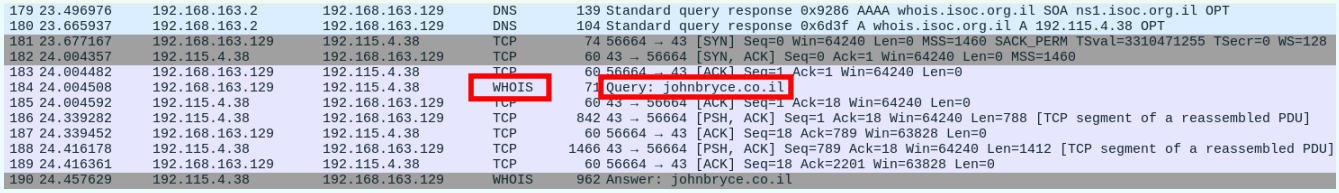
Tor commonly uses Port 9001 for network traffic and directory information. The sending and receiving of packets from Port 9001 shows that nipe connected to the Tor network successfully.



138	22.883193	192.168.163.128	192.168.163.129	SSHv2	98	Client: Protocol (SSH-2.0-OpenSSH_9.6p1 Debian-3)
139	22.883451	192.168.163.129	192.168.163.128	TCP	66	22 → 43378 [ACK] Seq=1 Ack=33 Win=65152 Len=0 TSval=328628574 TSecr=3107133956
140	22.891176	192.168.163.129	192.168.163.128	SSHv2	107	Server: Protocol (SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.6)
141	22.891206	192.168.163.128	192.168.163.129	TCP	66	43378 → 22 [ACK] Seq=33 Ack=42 Win=64256 Len=0 TSval=3107133964 TSecr=328628582
142	22.892511	192.168.163.129	192.168.163.128	SSHv2	1178	Server: Key Exchange Init
143	22.892526	192.168.163.128	192.168.163.129	TCP	66	43378 → 22 [ACK] Seq=33 Ack=1154 Win=64128 Len=0 TSval=3107133965 TSecr=328628583
144	22.899425	192.168.163.128	192.168.163.129	SSHv2	1602	Client: Key Exchange Init
145	22.899694	192.168.163.129	192.168.163.128	TCP	66	22 → 43378 [ACK] Seq=1154 Ack=1569 Win=64128 Len=0 TSval=328628590 TSecr=3107133972
146	22.963267	192.168.163.128	192.168.163.129	SSHv2	1274	Client: Diffie-Hellman Key Exchange Init
147	22.976176	192.168.163.129	192.168.163.128	SSHv2	1630	Server: Diffie-Hellman Key Exchange Reply, New Keys
148	22.976244	192.168.163.128	192.168.163.129	TCP	66	43378 → 22 [ACK] Seq=2777 Ack=2718 Win=64128 Len=0 TSval=3107134049 TSecr=328628667
149	23.000210	192.168.163.128	192.168.163.129	SSHv2	82	Client: New Keys
150	23.042030	192.168.163.129	192.168.163.128	TCP	66	22 → 43378 [ACK] Seq=2718 Ack=2793 Win=64128 Len=0 TSval=328628733 TSecr=3107134073
151	23.042065	192.168.163.128	192.168.163.129	SSHv2	110	Client:
152	23.042228	192.168.163.129	192.168.163.128	TCP	66	22 → 43378 [ACK] Seq=2718 Ack=2837 Win=64128 Len=0 TSval=328628733 TSecr=3107134115

Figure 10: packet capture for sshpass command

Port 22 is the default for SSH. The packet capture shows the Diffie-Hellman Key Exchange, a well-known public key cryptography method, that is commonly found when connecting via SSH.



179	23.496976	192.168.163.2	192.168.163.129	DNS	139	Standard query response 0x9286 AAAA whois.isoc.org.il SOA ns1.isoc.org.il OPT
180	23.665937	192.168.163.2	192.168.163.129	DNS	104	Standard query response 0x6d3f A whois.isoc.org.il A 192.115.4.38 OPT
181	23.677167	192.168.163.129	192.115.4.38	TCP	74	56664 → 43 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=3310471255 TSecr=0 WS=128
182	24.004357	192.115.4.38	192.168.163.129	TCP	60	43 → 56664 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
183	24.004482	192.168.163.129	192.115.4.38	TCP	60	56664 → 43 [ACK] Seq=1 Ack=1 Win=64240 Len=0
184	24.004508	192.168.163.129	192.115.4.38	WHOIS	71	Query: johnbryce.co.il
185	24.004592	192.115.4.38	192.168.163.129	TCP	60	43 → 56664 [ACK] Seq=1 Ack=18 Win=64240 Len=0
186	24.339282	192.115.4.38	192.168.163.129	TCP	842	43 → 56664 [PSH, ACK] Seq=1 Ack=18 Win=64240 Len=788 [TCP segment of a reassembled PDU]
187	24.339452	192.168.163.129	192.115.4.38	TCP	60	56664 → 43 [ACK] Seq=18 Ack=789 Win=63828 Len=0
188	24.416178	192.115.4.38	192.168.163.129	TCP	1466	43 → 56664 [PSH, ACK] Seq=789 Ack=18 Win=64240 Len=1412 [TCP segment of a reassembled PDU]
189	24.416361	192.168.163.129	192.115.4.38	TCP	60	56664 → 43 [ACK] Seq=18 Ack=2201 Win=63828 Len=0
190	24.457629	192.115.4.38	192.168.163.129	WHOIS	962	Answer: johnbryce.co.il

Figure 11: packet capture for whois scan on domain from user input

Port 43 is commonly used for whois protocol. The packet capture shows the scanning of the domain, johnbryce.co.il, from the user input.

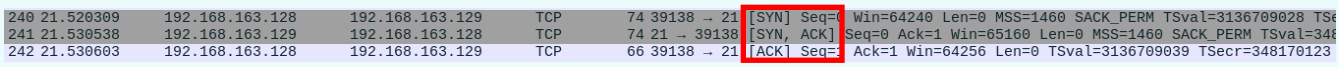


Figure 12: three-way handshake to establish TCP connection for FTP

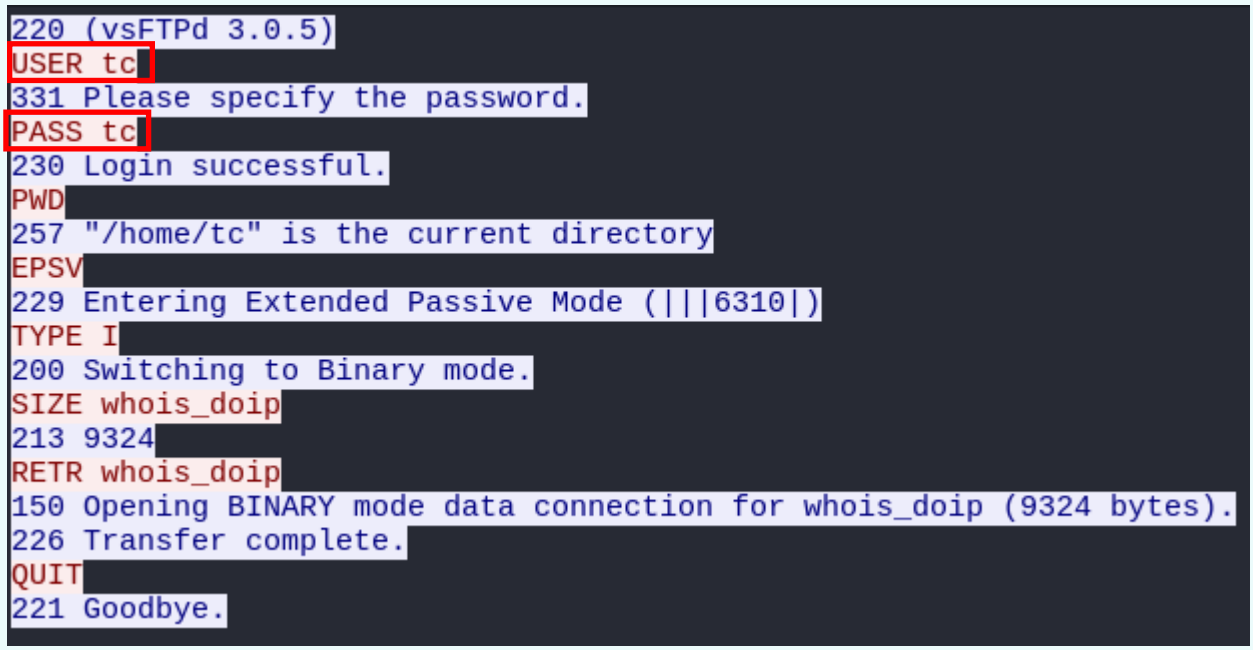


Figure 13: TCP Stream for FTP

FTP is commonly used on Port 21. The username and password for authentication of the file transfer via FTP can be seen clearly in plain text.

Analysis of network traffic for Secure File Transfer Protocol (SFTP)

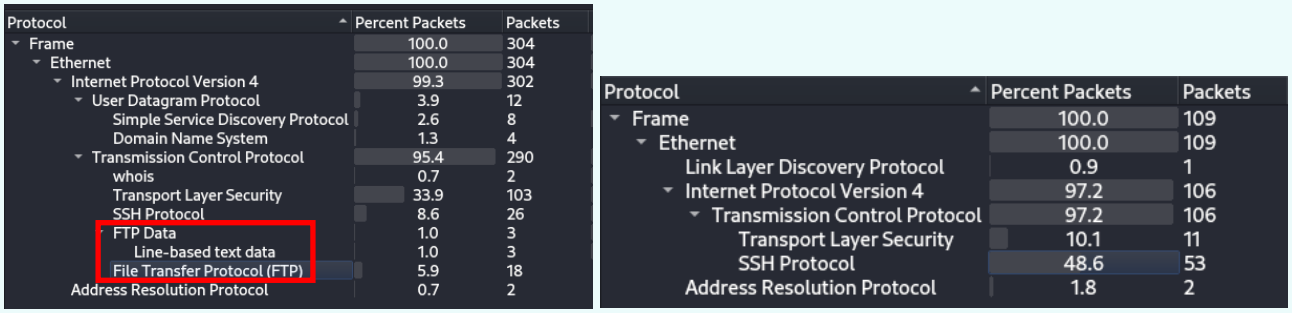


Figure 14: Protocol hierarchy for FTP (left) and SFTP (right)

FTP has its own protocol on WireShark, while SFTP transfers files over an encrypted SSH protocol, hence the lack of any SFTP protocols shown on Figure 11 (right). The username and password for authentication cannot be seen in the packets.

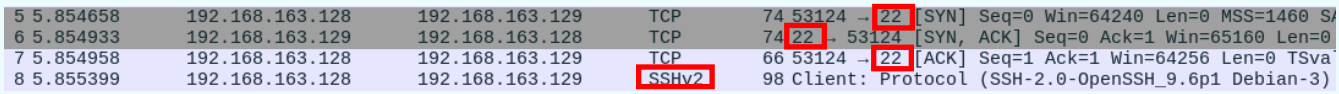


Figure 15: Establishing of SSH connection for SFTP on Port 22

Discussion

FTP

FTP is a standard network protocol used to transfer files between a client and a server on a network. The objectives of FTP are to promote sharing of files, encourage use of remote computers, shield users from variations in file storage systems and to transfer data reliably and efficiently.

FTP operates on a client-server architecture, where the sequence of events is as follows:

1) Establish control connection: A client first initiates a control connection over Port 21 to a server to perform file transfer operations.
2) Authentication: Users authenticate themselves to the FTP server using a username and password. The client provides authentication credentials to the server to gain access. However, some FTP servers can support anonymous authentication, allowing users to login with a generic username like "anonymous" and providing any password requested.
3) Command exchange and response: After authentication, users can use the client to send FTP commands to the server to perform various operations like listing directories, changing directories, uploading files, and downloading files. The server processes these commands and sends back responses indicating the success or failure of the operation. Every response the FTP server sends is preceded by a server return code.
4) Data transfer: During file transfer, FTP uses separate data connections for data transfer. The data connection is established either by the client (passive mode) or by the server (active mode). Data is then transferred over the data connection in accordance with the client's commands.
5) Session termination: After completion of file transfer, users may use the client to terminate the FTP session by sending a termination command to the server. The server acknowledges the termination request, and the control connection is closed.

Figure 16: Process of FTP file transfer

In the diagram, each step represents a specific interaction between the client and server during an FTP session, from connection establishment to session termination. The control connection handles command exchange and response handling, while a separate data connection is established for actual data transfer operations.

FTP also supports two types of operation for data transfer. ASCII transfer type is used for transferring text files, while binary transfer type is used for transferring non-text files.

FTP does not have standardized header structure like other protocols. The client usually sends one-word commands like ‘GET’ or ‘PUT’ and the server responds with a server return code and an optional response message to indicate the outcome of the requested operation.

Using the -A flag with ftp command forces the connection to be on active mode. By default, FTP uses passive mode and the client initiates the command channel, preventing the client firewall from blocking the FTP connection.

Advantages of FTP

FTP allows for the transfer of multiple files and directories simultaneously, ensuring that file transfers can be conducted faster.

There is no limit on the number of files or maximum file size of the files being transferred.

If connection is lost during file transfer, FTP can resume the interrupted file transfer.

FTP also supports scheduled transfers. This improves productivity as transfers can be scheduled during off-peak hours, with minimal impact on workflow.

FTP can be used by backup services or individual users to backup data from one location to a secured backup server.

Disadvantages of FTP

One major drawback for FTP is the lack of encryption of data. The username and password for authentication of the file transfer via FTP and the data transferred are sent in plain text, making FTP a very insecure protocol on its own. The data sent via FTP is vulnerable to sniffing, spoofing and brute force attacks.

FTP does not provide mechanisms for verifying the integrity of transferred files. Without integrity checks, there is a risk of files being tampered with or corrupted during transit.

Overall, FTP provides high availability due to its ease of setup and use. It has very useful features like transferring of multiple files and directories with no size restrictions.

FTP has low confidentiality and integrity due to the lack of encryption and verification of integrity. With the prevalence of cyber attackers in recent times, FTP is not a recommended method for file transfer. However, there are existing methods to make FTP more secure.

SFTP

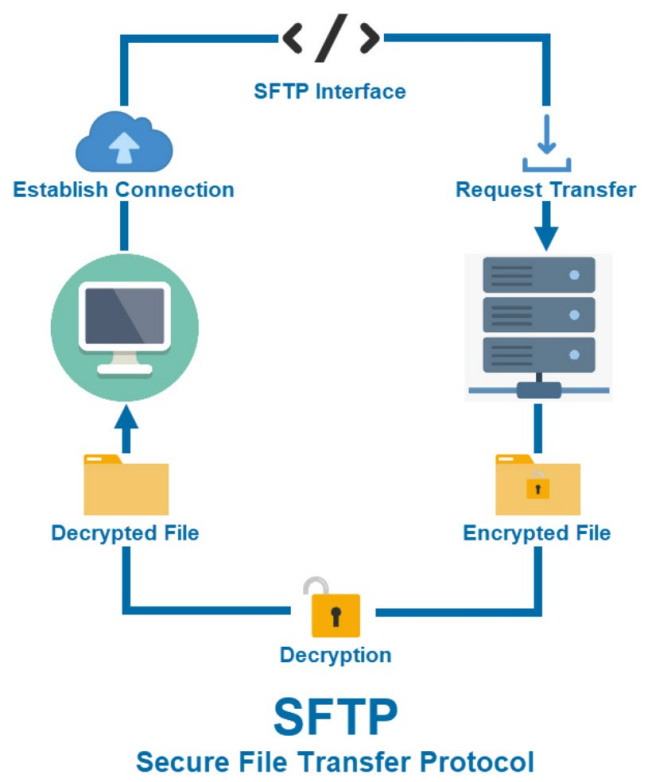


Figure 17: Process of SFTP file transfer

SFTP is a separate protocol that runs over SSH and provides secure file transfer capabilities. Unlike FTP, which operates over multiple ports, SFTP uses a single secure connection for both data transfer and command execution, simplifying firewall configurations and reducing security risks.

When compared with the traditional FTP protocol, SFTP offers all the functionality of FTP, but it is more secure.

SFTP uses an encryption cipher to encrypt the login credentials. Tunneling and public key authentication further protects the data. Hence, there is a higher degree of confidentiality compared to FTP.

SFTP does a SHA2 checksum on each packet sent. The receiver gets that packet and decrypts the data, and then verifies the checksum of the packet. This improves the integrity of data transfer using SFTP.

Despite the benefits of SFTP in improved security, the process of setting up and configuring servers and clients for SFTP is much more complex compared to FTP.

SFTP encryption and decryption processes can also consume more computational resources compared to unencrypted protocols like FTP.

In summary, while SFTP offers strong security features and compliance benefits, it will require more configuration and resources compared to traditional FTP. The benefits outweigh the disadvantages of using SFTP as securing data is critical in today's digital age.

Remote Control Bash Script

The purpose of the Remote Control Bash script is to help users automate work processes, hence reducing time required to complete certain processes (like scanning for domains of interest) and reduce mistakes through the use of the script.

The script also tries to improve security of the local machine by routing the network through the Tor network.

As the script was designed with the ease of accessibility in mind, the automation of the script was confined purely to the script, without the use of any configuration files and minimal input from the user. This has created a few security concerns.

Although sshpass uses SSH, which is a secure protocol, the automation consists of login credentials for the remote server being embedded within the script for both sshpass and FTP commands.

The condition in the script to check if nipe has been installed may not always be accurate, especially if nipe was only partially installed.

Conclusion

FTP has been a foundational protocol for file transfer over computer networks for several decades. Due to the lack of encryption, FTP relies on plaintext authentication, which is a vulnerability in the protocol. FTP also lacks verification of data integrity, which makes it susceptible to data corruption during transmission, and also man-in-the-middle attacks.

On the other hand, SFTP is an extension of SSH, providing secure file transfer capabilities over an encrypted connection. SFTP is also able to leverage on SSH's robust authentication mechanisms like the SSH keys, to better protect login credentials and enhancing security posture of the network. There is also cryptographic hash function utilized to ensure integrity in transferred files.

FTP alone is not suitable for transferring sensitive or confidential information in the current digital age. Organisations should consider adopting secure alternatives such as FTPS, SFTP, SCP, or HTTPS protocols to ensure improved confidentiality and integrity of their file transfer operations. This will also help organisations align with regulatory standards like the Personal Data Protection Act or Payment Card Industry Data Security Standard.

Therefore, SFTP is more suitable to be used in the Remote Control Bash script for better confidentiality and integrity.

The script's versatility and ease of use are among its main advantages. By accepting a few user-defined parameters, such as the remote server's IP address and the domain of interest, the script accommodates a wide range of data collection scenarios. This flexibility allows cybersecurity professionals to customise the script to their own requirements, whether it involves investigating potential security threats, analysing domain registrations, or monitoring online activity.

The automated nature of the script enhances operational efficiency and reduces the reliance on manual data collection techniques, which are inherently time-consuming and error-prone. By streamlining the data collection process, cybersecurity professionals can allocate their time and resources more effectively, focusing on critical analysis and response activities.

The routing of network through the Tor network provides anonymity and data encryption, allowing the script to provide better security to the cybersecurity professionals' machines, mitigating risks associated with loss of user data and better safeguarding the integrity of collected data.

In conclusion, the Remote Control Bash script represents a valuable tool after applying some enhancements like the use of SFTP and SSH key authentication. The script enables efficient and anonymous data collection from remote servers and facilitates the gathering of essential information for threat intelligence analysis.

Recommendations

Remote Control Bash Script

To improve security and reduce vulnerabilities in the script, the following methods can be used:

1) Store remote server login credentials in .netrc file and use the credentials for sshpass and SFTP. This will centralise each user's login credentials. Though the credentials are still stored in plaintext, the file contents can be more secure through restricting permissions.
2) The most secure way of automating SSH is to use SSH keys for authentication to replace password-based authentication. Private keys are stored in client's machine securely.
3) Certificate-based user authentication can be used along with SSH keys to enhance security.
4) Restrict the permissions of scripts containing sensitive information to provide a layer of access control and minimise security risks.
5) Implement input validation to prevent potential code injection attacks

The script could possibly check for the presence of key configuration files in the nipe folder to check if nipe has been properly installed. Alternatively, the script could try to run nipe and check for the status to determine if nipe is installed.

SFTP can be used instead of FTP to transfer files, ensuring better confidentiality and integrity. Another option is to use the script to install and run FTP applications like FileZilla that uses File Transfer Protocol Secure (FTPS) and SFTP.

The double quotation marks used after the sshpass command to ensure the subsequent commands are able to run on the remote server, makes it more difficult to edit the script. The use of '<< EOF' could also allow the script to enter a multi-line string until the tag of 'EOF'. It would be potentially easier to make adjustments to the script using '<< EOF' instead of the double quotation marks.

Providing training on the usage of the script will help users to use the script more effectively.

Updating Bash to the latest version would harden the security of the script.

Using the logger utility can also allow execution of the script to be logged into the syslog file, which allows for threat detection and digital forensics.

References

1. “Logo.” Bash Logo Media Assets, bashlogo.com/. Accessed 21 Mar. 2024.

2. Srivastava, Anurag. “Execute Commands on Remote Machines Using Sshpass.” Medium, 22 Aug. 2020, levelup.gitconnected.com/execute-commands-on-remote-machines-using-sshpass-1f9bc4452e15. Accessed 21 Mar. 2024.

3. “How Do You Escape Apostrophe in Single Quoted String in Bash?” Super User, superuser.com/questions/481797/how-do-you-escape-apostrophe-in-single-quoted-string-in-bash. Accessed 21 Mar. 2024.

4. “Use CURL and Wget to Download Network Files from CLI | TechTarget.” Networking, www.techtarget.com/searchnetworking/tutorial/Use-cURL-and-Wget-to-download-network-files-from-CLI.

5. “How to Grep for Contents after Pattern?” Stack Overflow, stackoverflow.com/questions/10358547/how-to-grep-for-contents-after-pattern. Accessed 21 Mar. 2024.

6. Networking. (n.d.). How to capture and analyze traffic with tcpdump | TechTarget. [online] Available at: https://www.techtarget.com/searchnetworking/tutorial/How-to-capture-and-analyze-traffic-with-tcpdump. Accessed 21 Mar. 2024.

7. “Rfc959.” Datatracker.ietf.org, datatracker.ietf.org/doc/html/rfc959.

8. EDUCBA. (2019). FTP vs SFTP | Top 12 Differences You Should Know. [online] Available at: https://www.educba.com/ftp-vs-sftp/. Accessed 21 Mar. 2024.

9. “What Is Anonymous FTP (File Transfer Protocol) and How Does It Work? – TechTarget Definition.” WhatIs.com, www.techtarget.com/whatis/definition/anonymous-FTP-File-Transfer-Protocol. Accessed 21 Mar. 2024.

10. “How Is the Integrity of Files Verified When Passed over SFTP?” Inspire-Tech, Inspire-Tech, support.inspire-tech.com/hc/en-us/articles/9740594773657-How-is-the-integrity-of-files-verified-when-passed-over-SFTP. Accessed 21 Mar. 2024.

11. “What Is SFTP? (Secure File Transfer Protocol).” Knowledge Base by PhoenixNAP, 25 Nov. 2021, phoenixnap.com/kb/what-is-sftp. Accessed 21 Mar. 2024.

12. Nigam, Pankhuri. “SSH, SSHPass and Passwordless SSH.” Medium, 19 Sept. 2018, medium.com/@pankhurinigam_47204/ssh-sshpass-and-passwordless-ssh-921210de0eb6. Accessed 21 Mar. 2024.