Unit: CFC011023

Network Research: Bash Scripting Automation & Network Protocol Research

Student No: S12

Student Name: Ivan Wong Sen Loong

Trainer’s Name: Mr. Tushar

Date of completion: 20-12-2023

Table of Contents

1.0 Introduction

The aim of this project is to develop a comprehensive understanding of network security and automation through two distinct scopes. Scope 1 focuses on creating an automated system for executing remote scripts, ensuring network anonymity, conducting remote server scans, and handling results securely. On the other hand, scope 2 involves manual network traffic capturing, analysis of unsecured protocols, and proposing secure alternatives. The project will be conducted on a clean Kali Linux environment and a remote server, utilizing tools like ssh, nipe, Nmap, whois, tcpdump, and Wireshark. The implementation will adhere to automation principles, using functions for streamlined processes and proper documentation through comments and a formal report structure.

2.0 Methodology

Scope 1: Bash Scripting from the local machine

The objective of the script is to automate the setup of an anonymous network connection using NIPE, perform network scanning using nmap, and execute remote commands on a specified IP address using SSH.

The methods used in the bash script to achieve the above are as follow:

Variables:

The script employs key variables such as **NIPE_DIR** for managing the NIPE directory path, **LOG_FILE** to specify the log file location, and **Spoof_IP** to capture the spoofed IP from NIPE status. The variable **Country_IP** determines the country of origin using **geoiplookup**. **log_dir** simplifies log directory management, while **user_IP** dynamically accepts user input for the target IP. **ssh_password** securely captures the SSH password.

Functions:

Central to the script's logging mechanism is the **log_message()** function. This function ensures systematic logging with timestamps, preventing duplicate entries. Its modular design enhances script readability and maintainability, contributing to a clear and organized record of execution steps.

Logging:

The script utilizes the **log_message()** function to timestamp and log execution steps. This approach provides a detailed and organized record in the specified log file, aiding in auditing, troubleshooting, and historical analysis. Systematic logging enhances the script's transparency and reliability.

And then a series of execution in the bash script:

Firstly on the **preparation phase**, the script begins by checking for root privileges to ensure that it has the necessary permissions to perform system-level tasks. It then verifies the existence of the log file directory and creates it if it does not already exist. If the log file itself is absent, the script creates it. To maintain a record of script execution, logging to syslog is enabled.

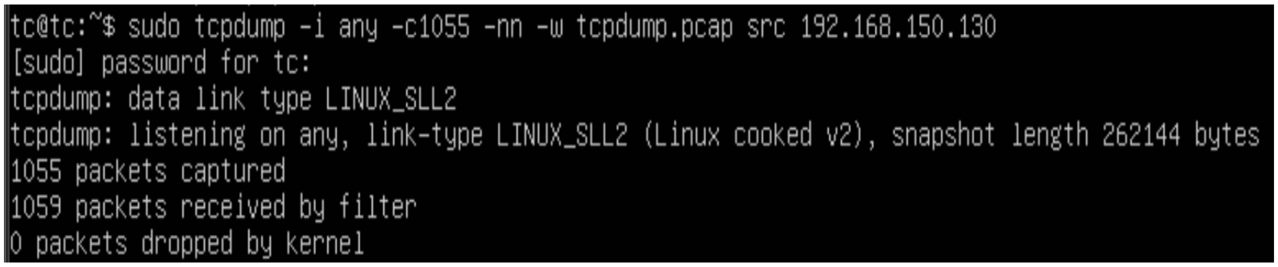
Moving on to the **NIPE installation phase**, the script checks for the presence of the NIPE directory. If NIPE is not installed, the script proceeds to install it. Additionally, the script ensures that the required Perl modules for NIPE are installed.

Next, in the **network anonymity phase**, the script checks if NIPE is already running to provide an anonymous network connection. If NIPE is not running, the script starts it to enable anonymous network communication.

During the **user interaction phase**, the script prompts the user to input the target IP address. It then initiates network reconnaissance using nmap on the specified IP to gather information about open ports and services running on the remote system.

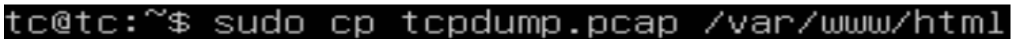
In the **SSH execution phase**, the script captures the SSH password securely from the user. It then utilizes SSH to execute remote commands on the specified IP, capturing information such as the IP address, country, and uptime. Additionally, the script retrieves the whois.txt file from the remote server using wget and copies the log file to the local directory for further analysis and record-keeping.

Scope 2: Capturing the packets from remote server



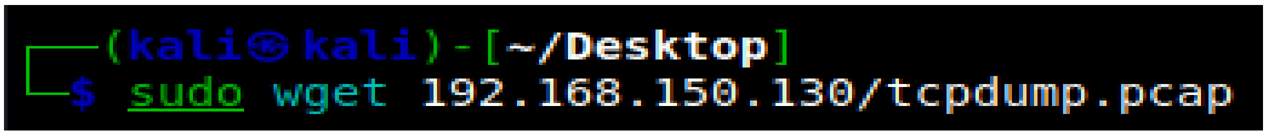
```
tc@tc:~$ sudo tcpdump -i any -c1055 -nn -w tcpdump.pcap src 192.168.150.130
[sudo] password for tc:
tcpdump: data link type LINUX_SLL2
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
1055 packets captured
1059 packets received by filter
0 packets dropped by kernel
```

The command “sudo tcpdump -i any -c1050 -nn -w tcpdump.pcap src 192.168.150.130” is then used in the remote server to capture incoming packets and put them into a .pcap file which will then be used to analyze later.



```
tc@tc:~$ sudo cp tcpdump.pcap /var/www/html
```

After tcpdump.pcap has been created, “sudo cp tcpdump.pcap /var/www/html” command is then used to store tcpdump.pcap onto the apache server.



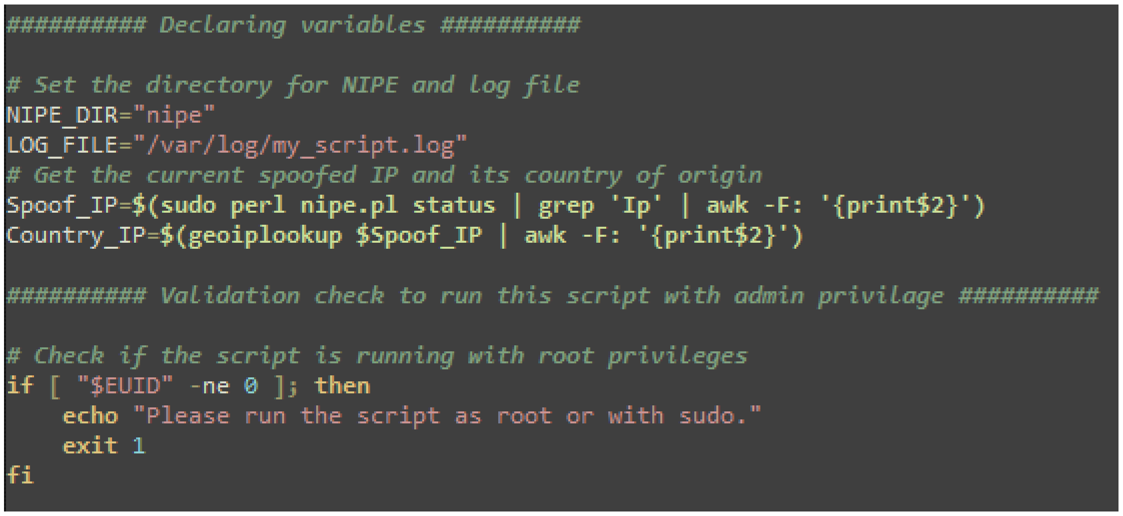
```
┌──(kali㉿kali)-[~/Desktop]
└─$ sudo wget 192.168.150.130/tcpdump.pcap
```

The above command is then used in the local machine to obtain the .pcap file.

3.0 Discussion

Scope 1: Bash Scripting Explanation

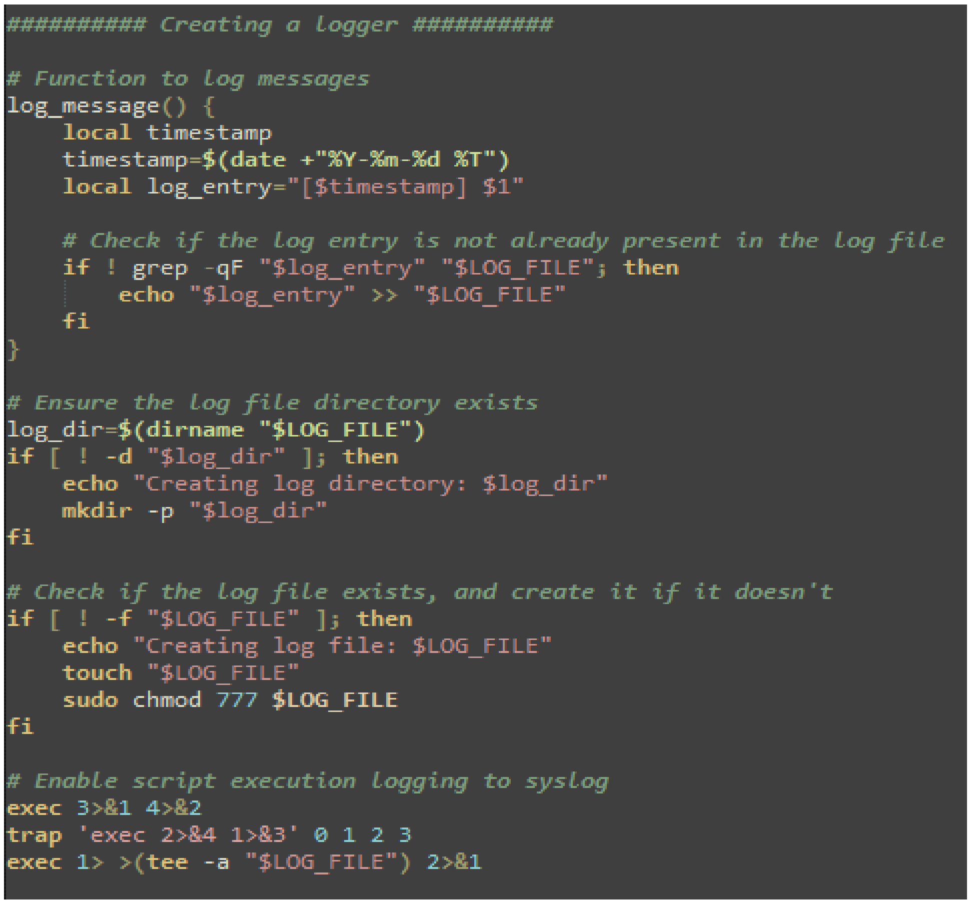
Preparation



```
########## Declaring variables ##########

# Set the directory for NIPE and log file
NIPE_DIR="nipe"
LOG_FILE="/var/log/my_script.log"
# Get the current spoofed IP and its country of origin
Spoof_IP=$(sudo perl nipe.pl status | grep 'Ip' | awk -F: '{print$2}')
Country_IP=$(geoiplookup $Spoof_IP | awk -F: '{print$2}')

########## Validation check to run this script with admin privilage ##########

# Check if the script is running with root privileges
if [ "$EUID" -ne 0 ]; then
    echo "Please run the script as root or with sudo."
    exit 1
fi
```

The script begins by setting up some essential variables. These include the directory where the NIPE tool is located **(`NIPE_DIR`)** and the path for the log file **(`LOG_FILE`)**. Additionally, it retrieves the current spoofed IP address by running a command using **`sudo`** to execute a Perl script called `nipe.pl`. It then uses the `geoiplookup` command to determine the country of origin for the spoofed IP address.

Before proceeding further, the script first checks whether it has been executed with administrative privileges. The **`[ "$EUID" -ne 0 ]`** holds the effective user ID, and a value of 0 indicates that the script is running as the root user. If the condition evaluates to true, meaning the user does not have root privileges, the script displays a message using `**echo**` instructing the user to run the script as root or with **`sudo`**. This check is important because certain operations, such as installing software or accessing system directories, require administrative permissions. If the script detects that it is not being run with the necessary privileges, it outputs a message asking the user to run it with **`sudo`** or as the root user, and then exits to prevent potential issues with incomplete or unauthorized operations.



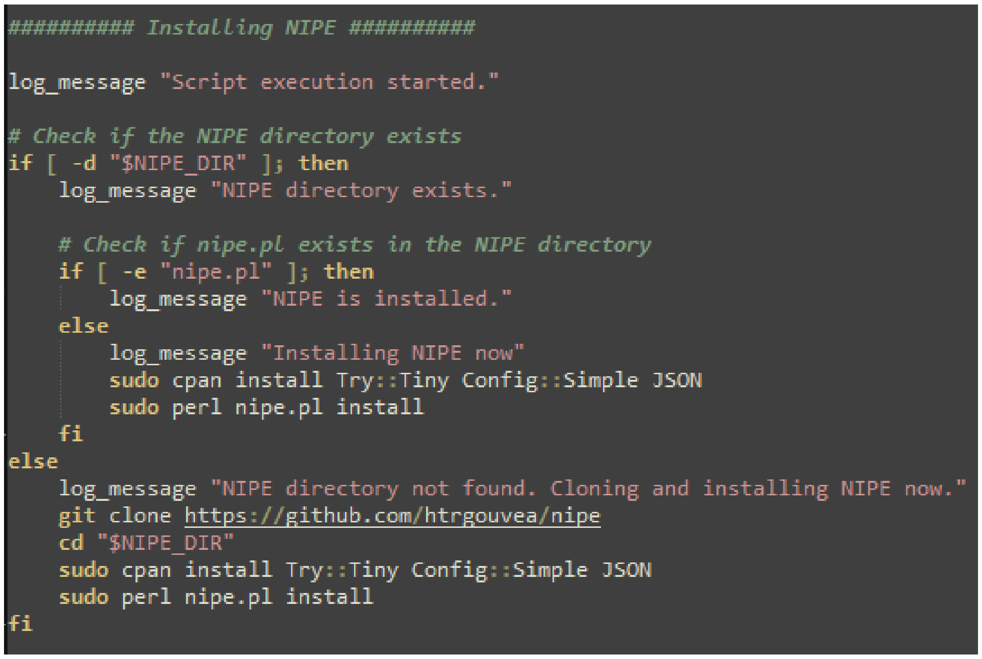
```
########## Creating a logger ##########

# Function to log messages
log_message() {
    local timestamp
    timestamp=$(date +"%Y-%m-%d %T")
    local log_entry="[$timestamp] $1"

    # Check if the log entry is not already present in the log file
    if ! grep -qF "$log_entry" "$LOG_FILE"; then
        echo "$log_entry" >> "$LOG_FILE"
    fi
}

# Ensure the log file directory exists
log_dir=$(dirname "$LOG_FILE")
if [ ! -d "$log_dir" ]; then
    echo "Creating log directory: $log_dir"
    mkdir -p "$log_dir"
fi

# Check if the log file exists, and create it if it doesn't
if [ ! -f "$LOG_FILE" ]; then
    echo "Creating log file: $LOG_FILE"
    touch "$LOG_FILE"
    sudo chmod 777 $LOG_FILE
fi

# Enable script execution logging to syslog
exec 3>&1 4>&2
trap 'exec 2>&4 1>&3' 0 1 2 3
exec 1> >(tee -a "$LOG_FILE") 2>&1
```

The script sets up a logging mechanism to record important messages during its execution. It defines a function called `**log_message()**` that takes a message as input and adds a timestamp to it. This function then checks if the combination of timestamp and message is already present in the log file. If not, it appends the message with its timestamp to the log file.

Before starting the logging process, the script ensures that the directory where the log file will be stored exists. If the directory doesn't exist, the script creates it using **`mkdir -p`**. Next, it checks if the log file itself exists. If not, it creates an empty log file using **`touch`** and sets its permissions to allow writing by any user using **`sudo chmod 777`**.

To enable script execution logging, the script uses file descriptors to redirect the standard output and error streams to the log file. It creates copies of the original standard output and error streams (file descriptors 3 and 4, respectively), then redirects the current standard output to a command that appends its input to the log file using **`tee`**. This allows the script to both display messages to the terminal and write them to the log file simultaneously.

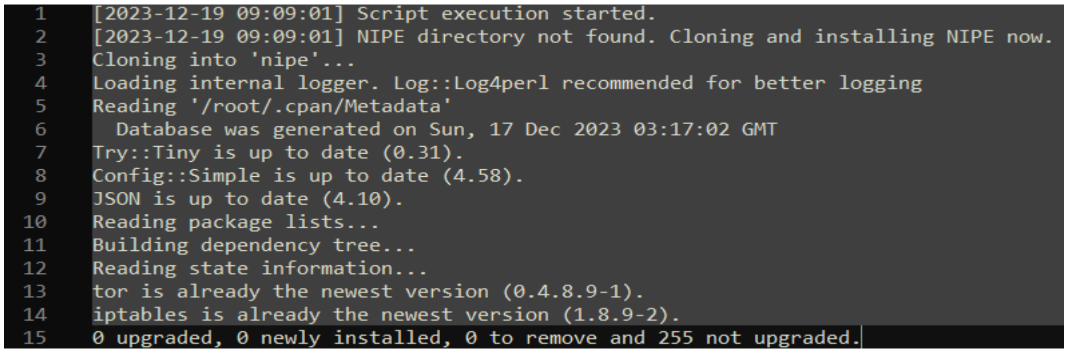
NIPE Installation



```
########## Installing NIPE ##########

log_message "Script execution started."

# Check if the NIPE directory exists
if [ -d "$NIPE_DIR" ]; then
    log_message "NIPE directory exists."

    # Check if nipe.pl exists in the NIPE directory
    if [ -e "nipe.pl" ]; then
        log_message "NIPE is installed."
    else
        log_message "Installing NIPE now"
        sudo cpan install Try::Tiny Config::Simple JSON
        sudo perl nipe.pl install
    fi
else
    log_message "NIPE directory not found. Cloning and installing NIPE now."
    git clone https://github.com/htrgouvea/nipe
    cd "$NIPE_DIR"
    sudo cpan install Try::Tiny Config::Simple JSON
    sudo perl nipe.pl install
fi
```

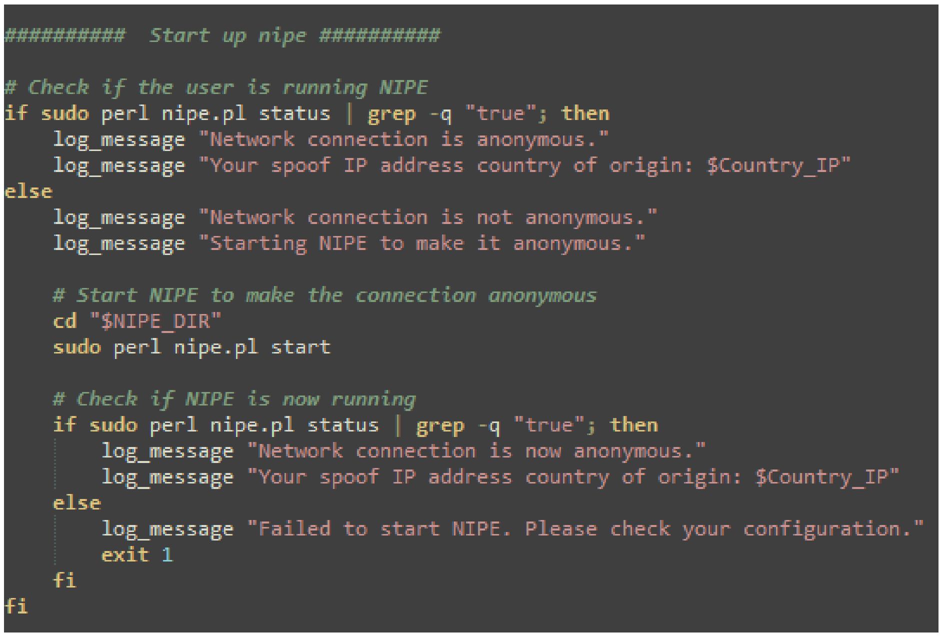
This portion of the script focuses on installing and configuring NIPE, a tool used for network anonymity. To begin, the script logs a message indicating the start of script execution. Following this, it checks if the NIPE directory (`NIPE_DIR`) exists. If the directory is found, the script logs a message confirming its existence. Within the NIPE directory, the script then checks if the `nipe.pl` script, a crucial component of NIPE, exists. If it does, the script logs a message indicating that NIPE is already installed. However, if **`nipe.pl`** is not found, the script logs a message stating that it's installing NIPE. The installation process involves using the **`sudo cpan install`** command to install necessary Perl modules **(`Try::Tiny`, `Config::Simple`, and `JSON`)**. Finally, the script executes the **`perl nipe.pl install`** command to complete the NIPE installation. If the NIPE directory is not found initially, the script logs a message indicating that it's cloning the NIPE repository from GitHub, then proceeds with the installation steps like those outlined above.

Output:



```
[2023-12-19 09:09:01] Script execution started.
[2023-12-19 09:09:01] NIPE directory not found. Cloning and installing NIPE now.
Cloning into 'nipe'...
Loading internal logger. Log::Log4perl recommended for better logging
Reading '/root/.cpan/Metadata'
  Database was generated on Sun, 17 Dec 2023 03:17:02 GMT
Try::Tiny is up to date (0.31).
Config::Simple is up to date (4.58).
JSON is up to date (4.10).
Reading package lists...
Building dependency tree...
Reading state information...
tor is already the newest version (0.4.8.9-1).
iptables is already the newest version (1.8.9-2).
0 upgraded, 0 newly installed, 0 to remove and 255 not upgraded.
```

Network Anonymity

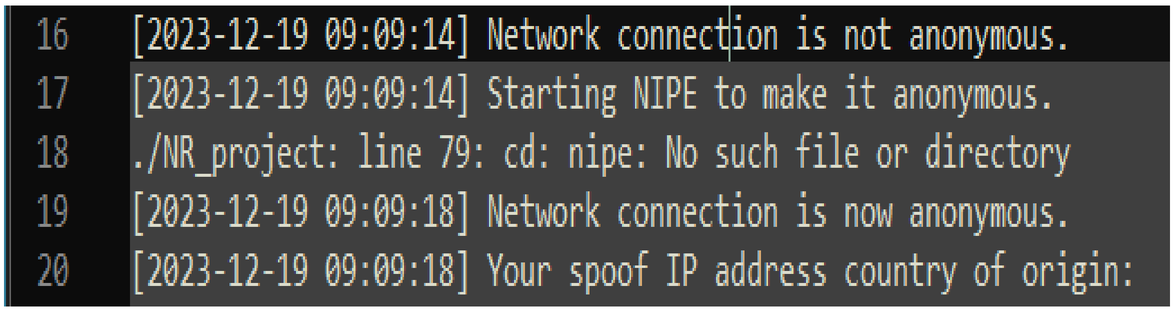


```
##########  Start up nipe ##########

# Check if the user is running NIPE
if sudo perl nipe.pl status | grep -q "true"; then
    log_message "Network connection is anonymous."
    log_message "Your spoof IP address country of origin: $Country_IP"
else
    log_message "Network connection is not anonymous."
    log_message "Starting NIPE to make it anonymous."

    # Start NIPE to make the connection anonymous
    cd "$NIPE_DIR"
    sudo perl nipe.pl start

    # Check if NIPE is now running
    if sudo perl nipe.pl status | grep -q "true"; then
        log_message "Network connection is now anonymous."
        log_message "Your spoof IP address country of origin: $Country_IP"
    else
        log_message "Failed to start NIPE. Please check your configuration."
        exit 1
    fi
fi
```

This section of the script checks whether NIPE, the tool used for network anonymity, is currently running. It does so by executing the command **`sudo perl nipe.pl status`** and checking its output for the presence of the string "true" using `**grep**`. If NIPE is found to be running, the script logs a message confirming that the network connection is anonymous, along with the country of origin of the spoofed IP address, which was previously determined and stored in the **`Country_IP`** variable.

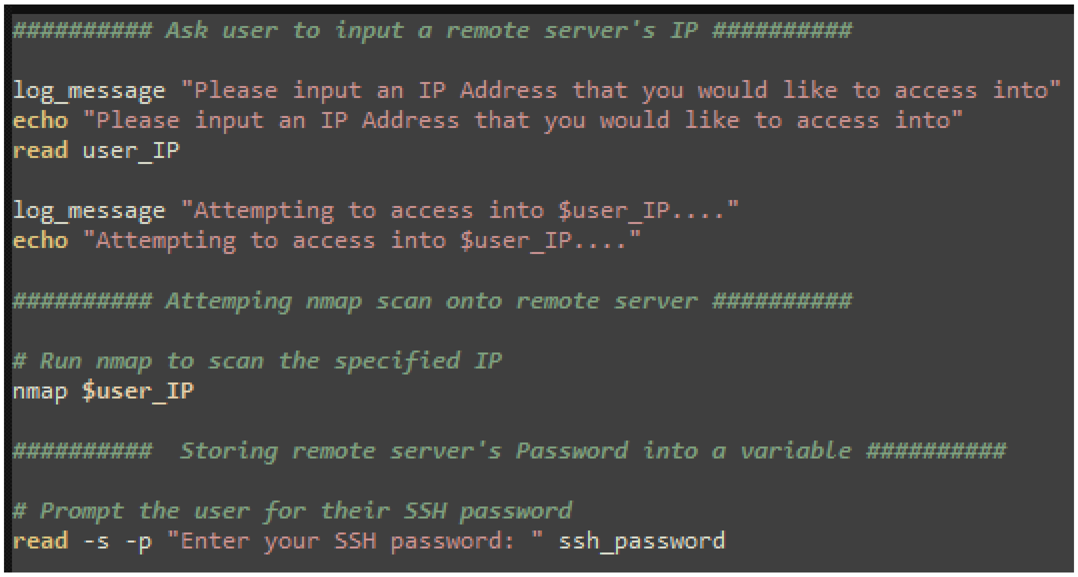
If NIPE is not running, the script logs a message indicating that the network connection is not anonymous and proceeds to start NIPE using the command **`sudo perl nipe.pl start`**. After starting NIPE, the script checks again if NIPE is now running by repeating the earlier status check. If NIPE is running after the start attempt, the script logs a message confirming the network connection is now anonymous, along with the country of origin of the spoofed IP address. However, if NIPE fails to start, the script logs an error message indicating the failure and exits with a status code of 1, signaling an unsuccessful operation.

Output:



```
[2023-12-19 09:09:14] Network connection is not anonymous.
[2023-12-19 09:09:14] Starting NIPE to make it anonymous.
./NR_project: line 79: cd: nipe: No such file or directory
[2023-12-19 09:09:18] Network connection is now anonymous.
[2023-12-19 09:09:18] Your spoof IP address country of origin:
```

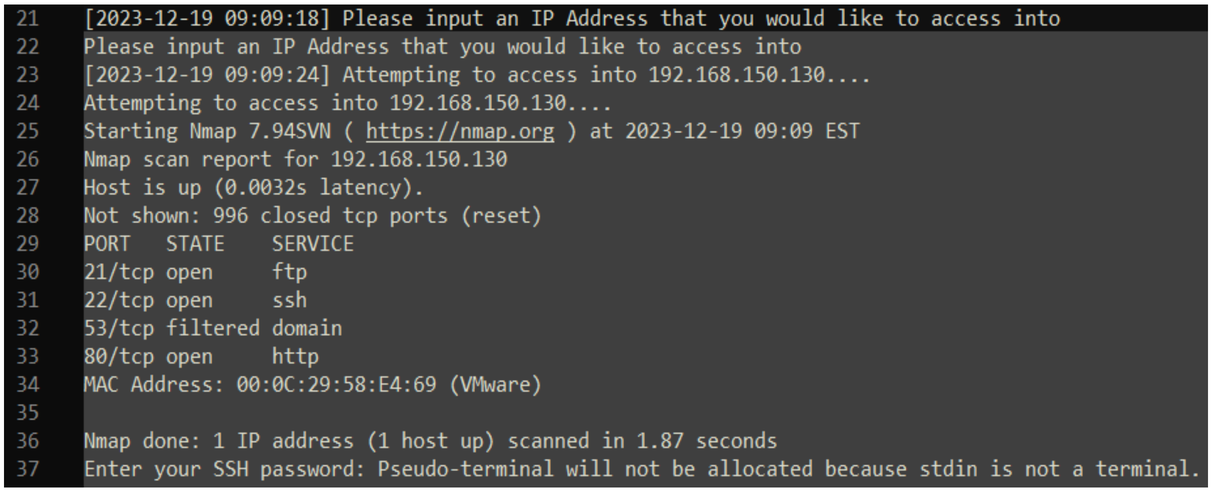
User Interaction



```
########## Ask user to input a remote server's IP ##########

log_message "Please input an IP Address that you would like to access into"
echo "Please input an IP Address that you would like to access into"
read user_IP

log_message "Attempting to access into $user_IP...."
echo "Attempting to access into $user_IP...."

########## Attemping nmap scan onto remote server ##########

# Run nmap to scan the specified IP
nmap $user_IP

##########  Storing remote server's Password into a variable ##########

# Prompt the user for their SSH password
read -s -p "Enter your SSH password: " ssh_password
```

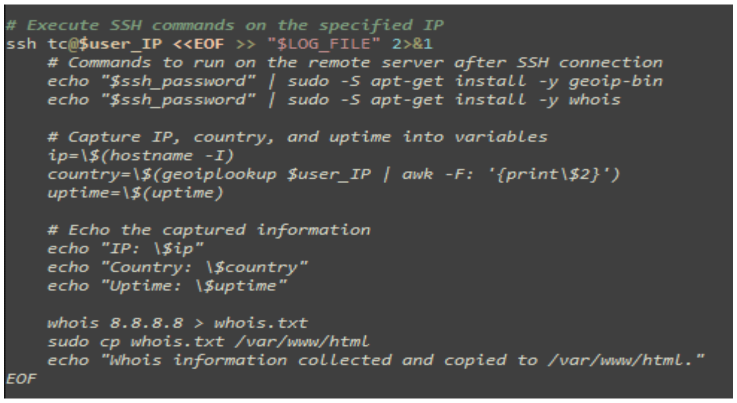
This part of the script prompts the user to input the IP address of a remote server they wish to access. It begins by logging a message, asking the user to input the IP address. Then, it displays a message on the terminal, <u>requesting the user to enter the IP address</u>. The script then reads the user's input and stores it in the variable `**user_IP**`. After capturing the user's input, the script logs a message indicating that it is attempting to access the server with the provided IP address. Next, the script uses the `**nmap**` command to perform a network scan on the specified IP address, which can provide information about open ports, services running, and other details about the network configuration of the remote server. Finally, <u>the script prompts the user to enter their SSH password securely</u>. The `**read**` command with the `**-s**` flag ensures that the password input is not displayed on the screen for security reasons. The entered password is stored in the variable `**ssh_password**` for later use in SSH authentication.

<u>Output:</u>



```
[2023-12-19 09:09:18] Please input an IP Address that you would like to access into
Please input an IP Address that you would like to access into
[2023-12-19 09:09:24] Attempting to access into 192.168.150.130....
Attempting to access into 192.168.150.130....
Starting Nmap 7.94SVN ( https://nmap.org ) at 2023-12-19 09:09 EST
Nmap scan report for 192.168.150.130
Host is up (0.0032s latency).
Not shown: 996 closed tcp ports (reset)
PORT   STATE    SERVICE
21/tcp open     ftp
22/tcp open     ssh
53/tcp filtered domain
80/tcp open     http
MAC Address: 00:0C:29:58:E4:69 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 1.87 seconds
Enter your SSH password: Pseudo-terminal will not be allocated because stdin is not a terminal.
```

SSH Execution



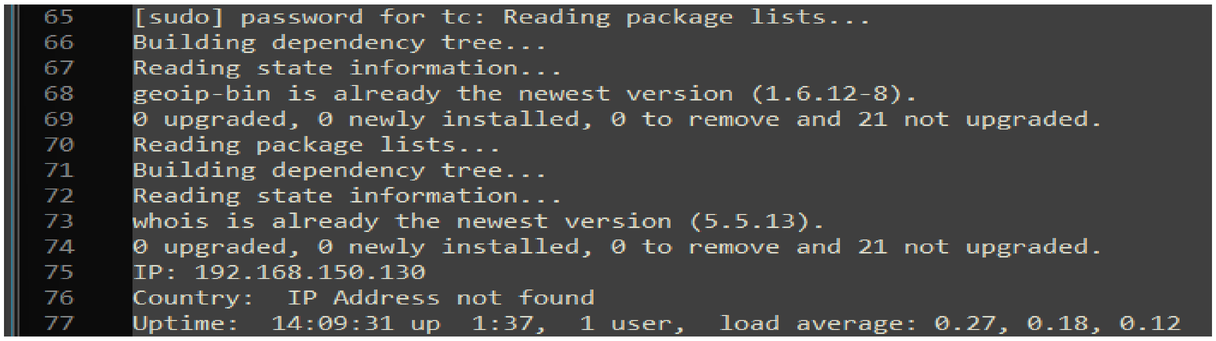
```
# Execute SSH commands on the specified IP
ssh tc@$user_IP <<EOF >> "$LOG_FILE" 2>&1
    # Commands to run on the remote server after SSH connection
    echo "$ssh_password" | sudo -S apt-get install -y geoip-bin
    echo "$ssh_password" | sudo -S apt-get install -y whois

    # Capture IP, country, and uptime into variables
    ip=\$(hostname -I)
    country=\$(geoiplookup $user_IP | awk -F: '{print\$2}')
    uptime=\$(uptime)

    # Echo the captured information
    echo "IP: \$ip"
    echo "Country: \$country"
    echo "Uptime: \$uptime"

    whois 8.8.8.8 > whois.txt
    sudo cp whois.txt /var/www/html
    echo "Whois information collected and copied to /var/www/html."
EOF
```

This part of the script establishes an SSH connection to the remote server using the `` `ssh` `` command with the specified username **(`tc`)** and IP address **(`$user_IP`)**. The **`<<EOF`** syntax is used to indicate the start of a here document, which allows multiple lines of input to be passed to the SSH session. The `` `EOF` `` at the end marks the end of the input.

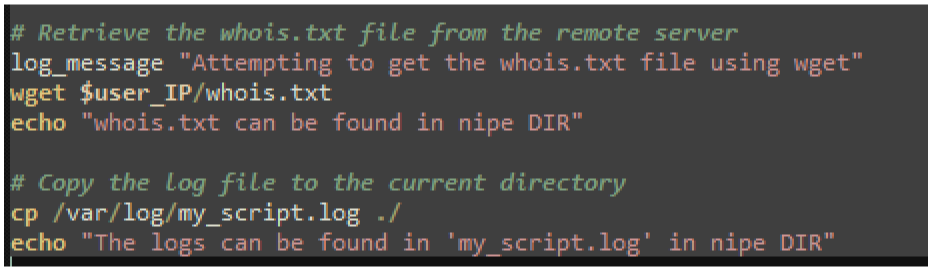
Within the SSH session, several commands are executed on the remote server. These commands are enclosed in the here document and include:

1.) Installing the **`geoip-bin`** and **`whois`** packages using **`apt-get`** with **`sudo`** and the previously captured **`ssh_password`**.
2.) Capturing the local IP address **(`ip`)**, country information **(`country`)**, and uptime **(`uptime`)** of the remote server into variables using various commands **(`hostname -I`, `geoiplookup`, and `uptime`)**.
3.) Echoing the captured information to display the IP address, country, and uptime on the terminal.
4.) Running **`whois 8.8.8.8`** to retrieve information about the IP address **`8.8.8.8`** and redirecting the output to a file named **`whois.txt`**.
5.) Copying the **`whois.txt`** file to the **`/var/www/html`** directory on the remote server using **`sudo cp`**.

Output:



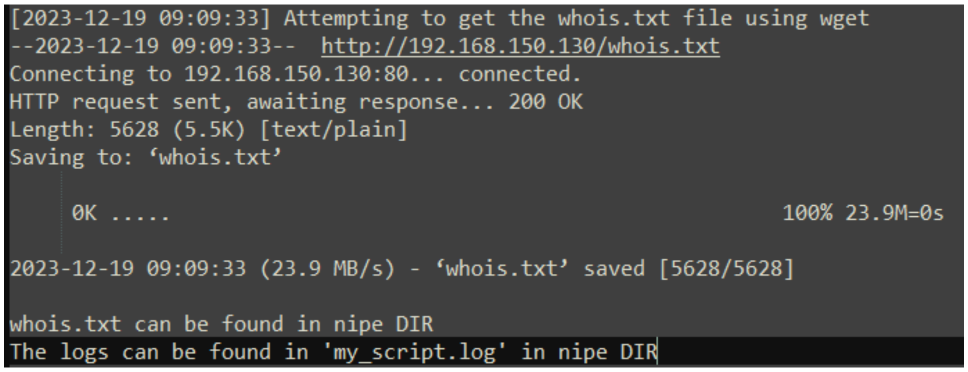
```
[sudo] password for tc: Reading package lists...
Building dependency tree...
Reading state information...
geoip-bin is already the newest version (1.6.12-8).
0 upgraded, 0 newly installed, 0 to remove and 21 not upgraded.
Reading package lists...
Building dependency tree...
Reading state information...
whois is already the newest version (5.5.13).
0 upgraded, 0 newly installed, 0 to remove and 21 not upgraded.
IP: 192.168.150.130
Country:  IP Address not found
Uptime:  14:09:31 up  1:37,  1 user,  load average: 0.27, 0.18, 0.12
```



```
# Retrieve the whois.txt file from the remote server
log_message "Attempting to get the whois.txt file using wget"
wget $user_IP/whois.txt
echo "whois.txt can be found in nipe DIR"

# Copy the log file to the current directory
cp /var/log/my_script.log ./
echo "The logs can be found in 'my_script.log' in nipe DIR"
```

After executing the commands on the remote server, the script attempts to retrieve the **`whois.txt`** file from the remote server using **`wget`**. It logs a message indicating this attempt and displays a message informing the user where to find the **`whois.txt`** file locally. Finally, it copies the log file **(`my_script.log`)** from the **`/var/log`** directory to the current directory and displays a message indicating where to find the log file.

Output:



```
[2023-12-19 09:09:33] Attempting to get the whois.txt file using wget
--2023-12-19 09:09:33--  http://192.168.150.130/whois.txt
Connecting to 192.168.150.130:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5628 (5.5K) [text/plain]
Saving to: ‘whois.txt’

     0K .....                                                 100% 23.9M=0s

2023-12-19 09:09:33 (23.9 MB/s) - ‘whois.txt’ saved [5628/5628]

whois.txt can be found in nipe DIR
The logs can be found in 'my_script.log' in nipe DIR
```

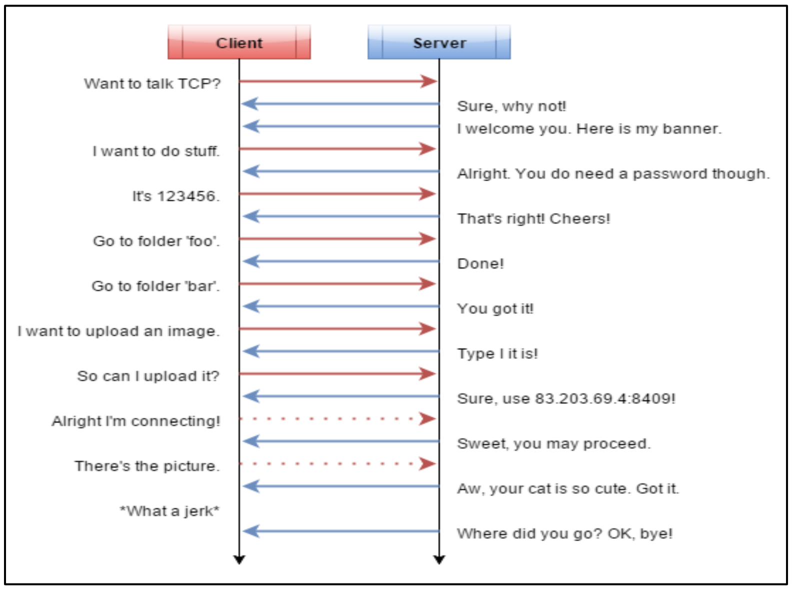
Scope 2: Wireshark Protocol Explanation: FTP

1. Understand the purpose, key features, and the problem FTP aims to solve. Gather resources such as RFCs (Request for Comments), protocol specifications, and relevant literature.

File Transfer Protocol (FTP) is a standard network protocol used for transferring files between a client and a server on a computer network. Its purpose is to provide a simple and efficient means of transferring files over a network, which is typically the Internet or sometimes through a GUI. Key features include support for various file transfer modes (for example ASCII, binary), user authentication, and directory listing. FTP aims to solve the problem of transferring files reliably and efficiently between different systems, regardless of their underlying architectures.

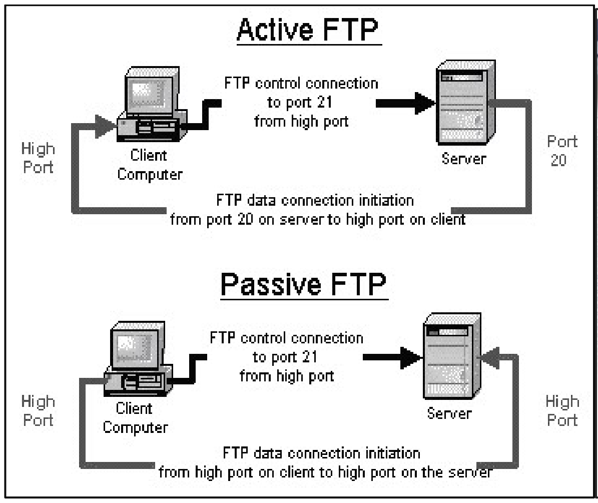
2. Describe the fundamental behaviors of FTP. Explain how it works, the message exchange process, and the sequence of events. Use diagrams and flowcharts to illustrate the protocol's behaviors

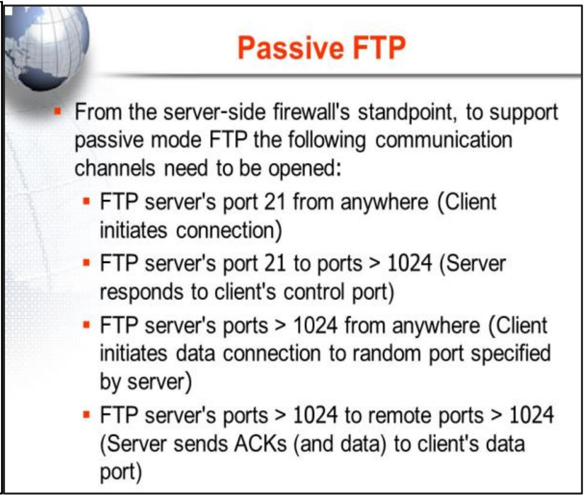
The fundamental behavior of FTP involves a client-server model, where the client initiates a connection to the server's port 21 for control information. Then, the client can then request actions from the server, such as listing directories, uploading files, or downloading files. The protocol uses a command-response mechanism, where commands are sent by the client and responses are sent by the server. The sequence of events typically involves authentication, data transfer commands (e.g., STOR for storing files, RETR for retrieving files), and closing the connection. Here is a layman diagram showing the protocol's behavior (with a hint of 3-way handshake in action)



3. Dive deeper into the FTP mechanisms. Explore aspects like header structure, message formats, and any flags or options that affect its behaviors.

FTP operates using a simple text-based protocol where commands and responses are exchanged between the client and the server. Commands are sent in the form of ASCII text strings, with each command ending with a carriage return and line feed. The protocol includes commands for <u>authentication, file operations, directory manipulation, and control operations</u>. The data transfer process can occur in two modes: **active mode**, where the server initiates the data connection, and **passive mode**, where the client initiates the data connection.


Active FTP
FTP control connection to port 21 from high port
High Port
Client Computer
Server
Port 20
FTP data connection initiation from port 20 on server to high port on client
Passive FTP
FTP control connection to port 21 from high port
High Port
Client Computer
Server
High Port
FTP data connection initiation from high port on client to high port on the server



Passive FTP

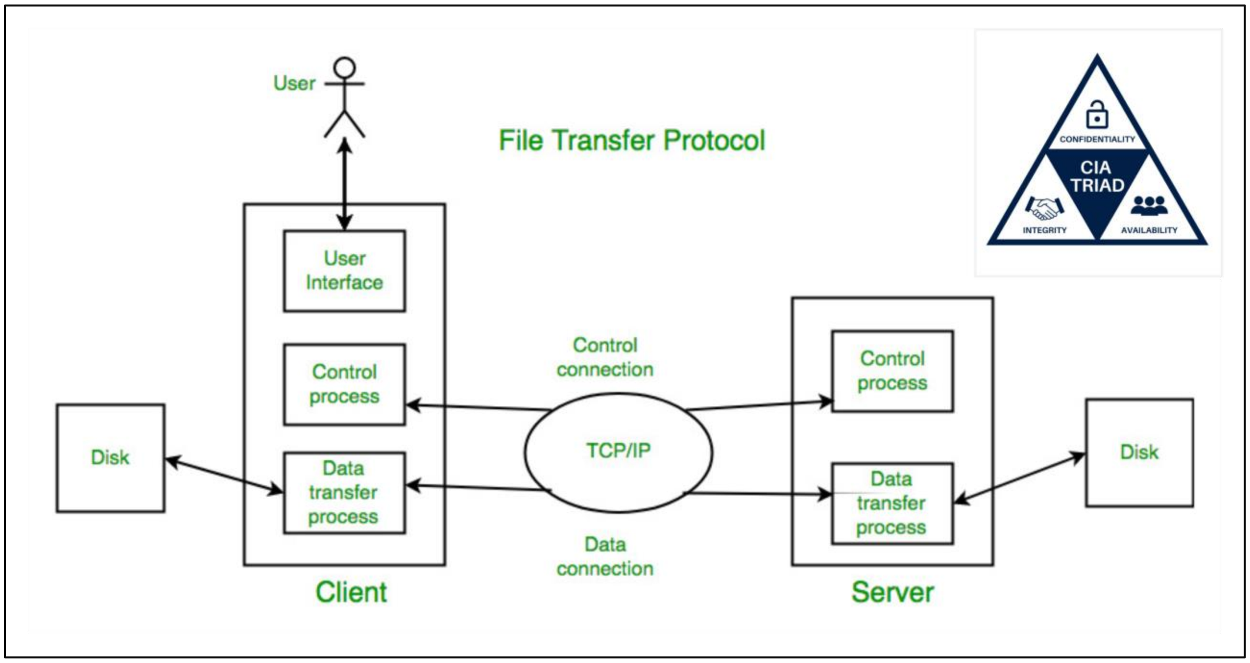
- From the server-side firewall's standpoint, to support passive mode FTP the following communication channels need to be opened:
  - FTP server's port 21 from anywhere (Client initiates connection)
  - FTP server's port 21 to ports > 1024 (Server responds to client's control port)
  - FTP server's ports > 1024 from anywhere (Client initiates data connection to random port specified by server)
  - FTP server's ports > 1024 to remote ports > 1024 (Server sends ACKs (and data) to client's data port)

4. Discuss the strengths and weaknesses of the protocol and relating it impacting the CIA Triad.

The strengths of FTP include its widespread support across different platforms and its simplicity, making it easy to implement and use. However, FTP has weaknesses related to security, as it does not encrypt data or credentials by default, making it vulnerable to eavesdropping and data tampering.

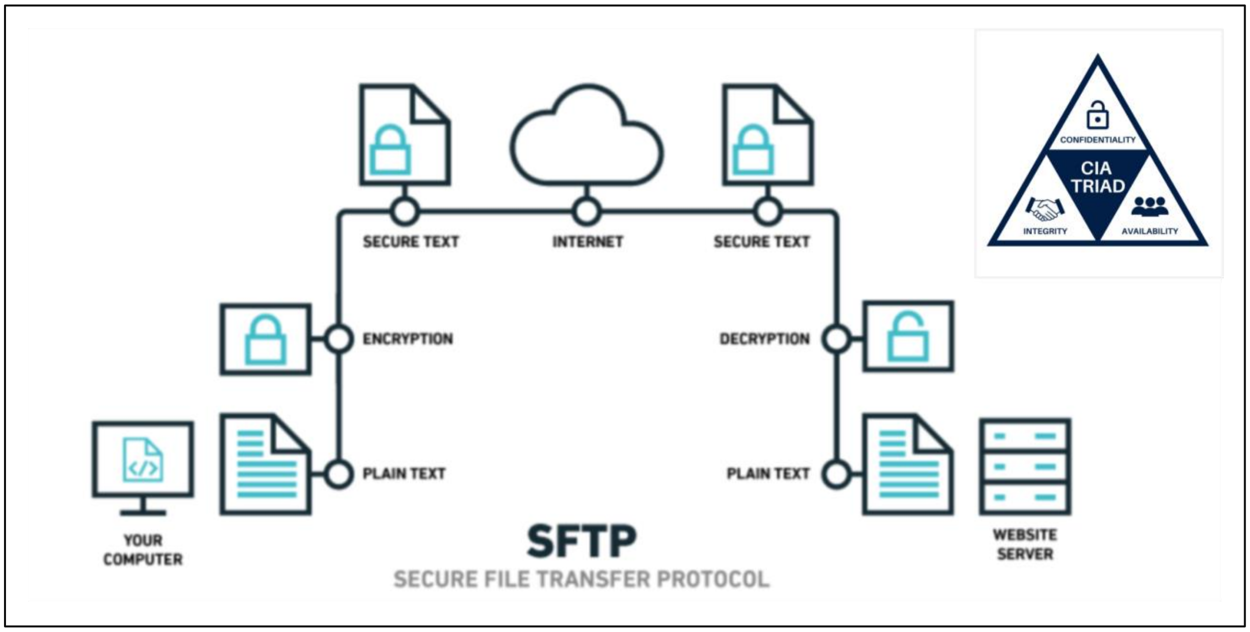
From a CIA Triad perspective (confidentiality, integrity, availability), FTP's lack of encryption impacts confidentiality, while its reliance on plaintext passwords and limited authentication options can affect both confidentiality and integrity. Additionally, FTP's use of separate control and data connections can lead to firewall and NAT traversal issues, affecting availability.

FTP and CIA Triad



Confidentiality	FTP **does not provide inherent encryption** for data transmission, which means that data transferred using FTP is susceptible to interception and eavesdropping.
Integrity	FTP **does not include built-in mechanisms for ensuring the integrity of transferred files.** Without mechanisms like cryptographic hash functions or digital signatures, there's a higher risk of files being modified or corrupted during transmission without detection.
Availability	FTP is often used for transferring files over networks, contributing to the availability of data. However, its **reliance on clear-text transmission and lack of robust security features** can make it vulnerable to attacks that could compromise availability, such as denial-of-service (DoS) attacks or eavesdropping

SFTP and CIA Triad



Confidentiality	SFTP provides a **secure channel** for file transfer, encrypting both the session and the data being transferred. This encryption ensures that data remains confidential and is protected from unauthorized access or interception during transmission.
Integrity	SFTP **employs mechanisms such as cryptographic hash functions** to verify the integrity of transferred files. This ensures that files remain unchanged and uncorrupted during transit, maintaining their integrity and reliability.
Availability	By **offering secure and reliable file transfer capabilities** (error-checking mechanisms, such as checksums), SFTP contributes to the availability of data. It ensures that authorized users can access and transfer files without disruption while protecting against unauthorized access that could compromise availability.

4.0 Conclusion

To wrap up everything in a nutshell, this project is a comprehensive exploration of network security, automation, and protocol analysis. Through Scope 1, I explored ways to construct an automated script which can execute remote scripts, ensuring network anonymity, and securely managing results. Subsequently in scope 2, I had delved into manual network traffic capture, protocol analysis, and proposed secure alternatives to address the CIA Triad's impact.

The project has made me realize the importance of knowing how to construct basic bash scripts to feed my needs and also enhanced my understanding of network protocols, their vulnerabilities, and the critical role of security in data transmission. Moving forward, it is essential to continue exposing myself in different bash scripts with different functions and to explore emerging security technologies and best practices to stay ahead of evolving threats in network environments.

5.0 Recommendation

Based on the findings and outcomes of this project, several recommendations can be made to further enhance network security and automation practices.

1. **Secure File Retrieval:** In my bash scripting I had used http request to obtain the file and, in the future, I would consider using Secure Copy Protocol (SCP) instead of FTP or HTTP for file retrieval from the remote server. SCP provides secure and encrypted file transfers over SSH, offering improved security compared to FTP or HTTP.

2. **Enhanced Spoofing Mechanism:** To enhance my anonymity, I would consider the use of VPNs (Virtual Private Networks) to enhance the spoofing mechanism in my scripts. It seems that VPNs can provide an additional layer of security by encrypting network traffic and masking the true origin of the connection.

3. **Improved Logging and Auditing:** To enhance the logging and auditing capabilities of my scripts I would implement a more comprehensive logging mechanism such as syslog-ng or a similar logging framework to centralize logs and improve visibility into script execution and network activities. This can help me monitor and detect any suspicious activities more effectively.

By incorporating these recommendations, I should be able to enhance the overall security posture and functionality of my network automation tasks while ensuring that they remain robust and resilient against potential threats.

6.0 Reference

RFC 959: File Transfer Protocol. (n.d.). IETF Datatracker. https://datatracker.ietf.org/doc/html/rfc959

Why Does FTP Suck? | Why Does It Suck? (n.d.). https://whydoesitsuck.com/why-does-ftp-suck/

Nag, S. (2021, May 28). *Building Protocol with SFTP*. https://www.linkedin.com/pulse/building-protocol-sftp-sibendu-nag-

How to Install Nipe tool in Kali Linux. (2021, October 5). GeeksforGeeks. https://www.geeksforgeeks.org/how-to-install-nipe-tool-in-kali-linux/

Kerner, S. M., & Burke, J. (2021, May 6). *FTP (File Transfer Protocol)*. Networking. https://www.techtarget.com/searchnetworking/definition/File-Transfer-Protocol-FTP

Gerardi, R. (n.d.). *An introduction to using tcpdump at the Linux command line*. Opensource.com. https://opensource.com/article/18/10/introduction-tcpdump