

CFC190324

(Network Research)

Project Title: Network Remote  
Control

Student Code: S8

Student Name: Jeremiah Lee

Trainer Name: Samson

# Table of Contents

1) <b>Introduction</b> .....	4
2) <b>Methodologies</b> .....	5
2.1) Installations and Anonymity Check.....	5
2.1.1) Checking for existing applications.....	5
2.1.2) Installing the missing applications.....	7
2.1.3) Applications already installed.....	7
2.1.4) Installing nipe.pl and checking anonymous status.....	7
2.1.5) Installing nipe.pl.....	8
2.1.6) Checking anonymous status.....	12
2.1.7) Allow the user to specify the address/URL to scan from the remote server.....	17
3.1) Automatically Scan the Remote Server for open ports.....	18
3.1.1) Connect to the Remote Server via SSH.....	18
3.1.2) Display the details of the remote server.....	19
4.1) Scan (NMAP and whois) the domain/url provided by the user on the remote server and save the results of the scan onto the remote server(Ubuntu).....	22
4.1.1) Changing permissions of /var and /var/log folder.....	22
4.1.2) Create a scan log folder to document scan details.....	23
4.1.3) whois/nmap scanning and saving into /var/log.....	24
4.1.4) Saving scanned details into scanned.log.....	25
4.1.5) Retrieve the results of the scan from your local machine.....	27
4.1.6) Capturing of traffic using tcpdump and wireshark.....	28
3) <b>Discussion</b> .....	29
3.1) Installations and anonymity check.....	29
3.2) Checking for nipe.pl.....	31
3.3) If anonymous, to display the spoof country name.....	34
3.4) Allow the user to specify the address/URL to whois from remote server; save into a variable.....	35
3.5) Connect to the remote server via ssh.....	35
3.6) Display the details of the remote server.....	36
3.7) Conduct an nmap and whois scan and save the details into the remote computer.....	37
3.8) Creating scanned.log folder and giving chmod 777 permissions for the scanned.log file.....	39
3.9) Scanning and saving the results of the scan into the remote server.....	39
3.10) Appending main details of scan into scanned.log file.....	40

4) Research of selected protocol (File Transfer Protocol).....	42
5) Suggested secure network protocol.....	50
6) Conclusion.....	52
7) Recommendations.....	53
References.....	56

# 1) Introduction

In today's dynamic cybersecurity landscape, the imperative to defend against ever-evolving threats has never been more critical. As malicious actors continuously adapt their tactics, it becomes essential to explore innovative strategies to safeguard digital assets and networks.

Our cybersecurity project focuses on two pivotal scopes designed to enhance network defences and proactive security measures: Network Remote Control and Network Research and Monitoring.

Our project's first scope focuses on the development of automation tools empowering cyber units to execute scripts from their local devices while harnessing the computational prowess of remote servers. This functionality fosters seamless communication with remote servers, enabling the execution of automated tasks with anonymity and efficiency. Leveraging remote servers' capabilities empowers cyber units to respond swiftly and discreetly to threats, enhancing their operational effectiveness.

In the second scope, we delve into the nuances of network research and monitoring, pivotal elements of proactive cybersecurity strategies. Here, our aim is to gain deeper insights into network architecture and security vulnerabilities by capturing and analysing traffic during automated attacks on servers. We seek to elucidate the vulnerabilities associated with unsecure protocols like FTP and their profound impact on the core tenets of the CIA Triad: confidentiality, integrity, and availability..

## 2) Methodologies

In this chapter, we will briefly describe the commands used to obtain the relevant information.

### Scope 1 (Automation)

#### 2.1) Installations and Anonymity Check

##### 2.1.1) Checking for existing applications

Before installing the needed applications, we needed to identify which of the needed applications were already in the system. For that we used the following command:

```
command -v <target command>
```

The command `command -v <target_command>` is specifically used to check for the availability and location of another command on your system. Here's a breakdown of its functionality:

**Command:** This is a built-in shell command that helps manage how other commands are executed.

**-v:** This option with `command` tells it to provide verbose information about the target command.

**<target\_command>:** This is the name of the command you want to check for.

The “`command -v <target command>`” was put in “if” command.

**If command:** An if command is a fundamental conditional statement used in shell scripting and command-line operations. It allows us to execute a certain block of code or command based on whether a specified condition evaluates to true or false.

```
if [ condition ]; then
    # Code to execute if the condition is true
else
    # Code to execute if the condition is false
fi
```

In this case, the “`command -v <target command>`” was put into a condition where by

It would equate to an output if the command was present and installed and would not create an output if the command was not present or installed.

For this script, the required commands were

- 1) geoiplookup
- 1) Sshpass
- 2) nipe.pl

```
if [ $(command -v geoiplookup) == -z ]
```

**-z:** In Linux, the -z condition in the if statement is used to check if a string is empty (has no length or output). It evaluates to true if the specified string is empty and false if it is not empty.

```
$(command -v geoiplookup) == -z ]
then  Nil output-> install (1.2)
      echo "geoiplookup is required but it
      sudo apt-get install geoiplookup #in
else  output detected -> no need install (1.3)
      echo "geoiplookup application is ins
```

### 2.1.2) Installing the missing applications

If the condition was met (i.e. there was no output detected/ == -z is true), then the Script will proceed to install the required files.

**Sudo apt-get install <command name>**

**sudo:** This command is used to execute another command as a superuser (or root user) in Linux. It stands for "superuser do." Superuser privileges are required to install software packages and make system-level changes.

**apt-get:** This is the package management tool used in Debian-based Linux distributions to handle the installation, removal, and upgrading of software packages. It stands for "Advanced Package Tool."

**install:** This is a sub-command of apt-get that specifies the action to be performed, in this case, installing a package.

**echo:** command in Linux is used to display text or variables on the terminal or in shell scripts.

### 2.1.3) Applications already installed

If no output is detected, the script proceeds without any additional commands done other than the echo command informing the user that the required application is already installed.

### 2.1.4) Installing nipe.pl and checking anonymous status

The mentioned method of checking if commands were present would not work for nipe.pl. This is because nipe.pl is a perl script, not a compiled executable (Perldoc, n.d.)

This implies that Perl scripts are not transformed into independent executable files, as seen in languages such as C or C++. Instead, they are interpreted by the Perl interpreter as they are run

Instead of using command -v, the following command was used to identify if nipe.pl was available.

## Locate nipe.pl

**Locate:** The command is used to find the location of files by searching a pre-built database of filenames and their paths. The command then prints out the paths of all files matching the specified pattern. By default, it prints the absolute paths of all matching files. If there is no matching filename or directory, no output will be printed out.

Similar to the method mentioned in 1.1, the if command was used with

```
locate nipe.pl == -z
```

Set as the condition.

Here is a refresher of said explanation.

**-z:** In Linux, the `-z` condition in the if statement is used to check if a string is empty (has zero length). It evaluates to true if the specified string is empty and false if it is not empty.

In this case, if `locate nipe` did not yield any output (i.e. does not exist), the script will proceed to install `nipe.pl` and its associated codes and applications. This will be explained in 1.4.1

If `locate nipe.pl` yielded an output (`nipe.pl` exists), then the script will check on its anonymous status. This will be explained in 1.4.2.

### 2.1.5) Installing `nipe.pl`

The following commands were used to install `nipe.pl`

```
git clone https://github.com/htrgouvea/nipe && cd nipe
```

The command provided is a combination of two separate commands joined by the `&&` operator.

**git clone https://github.com/htrgouvea/nipe:** This command clones a Git repository located at the specified URL (`https://github.com/htrgouvea/nipe`) onto our local machine. In this case, it clones the repository named "nipe" from the GitHub user "htrgouvea" to our current directory.

**git clone:** This is the Git command used to clone a repository.

`https://github.com/htrgouvea/nipe:` This is the URL of the Git repository we want to clone.

**&&:** This is a logical operator used to combine multiple commands in a single line. In this context, it means "execute the next command only if the previous command succeeds."

**cd nipe:** This command changes the current directory to the "nipe" directory that was just cloned from the Git repository. After successfully cloning the repository, the `cd` command moves us into the newly created "nipe" directory.

**cd:** This is the command used to change directories.



**nipe:** This is the name of the directory you want to change to.

When we run `git clone https://github.com/htrgouvea/nipe && cd nipe`, Git clones the "nipe" repository from GitHub, and if the cloning is successful, it changes our current directory to the "nipe" directory. This is often used to quickly clone a repository and navigate into it in one go.

## `sudo apt-get install cpanminus`

**sudo:** This command is used to execute another command with superuser privileges. Superuser privileges are required to install software packages and make system-level changes. By using `sudo`, we are indicating that we want to run the following command (`apt-get install cpanminus`) with elevated permissions.

**apt-get:** This is the command-line tool used for handling packages in Debian-based Linux distributions. It's used to install, remove, and manage software packages. In this case, `apt-get` is being used to install the `cpanminus` package.

**install:** This is a sub-command of `apt-get` that specifies the action to be performed, which in this case is the installation of a package.

**cpanminus:** This is the name of the package that you want to install. `cpanminus` is a Perl module installer that makes it easier to manage Perl modules from CPAN (Comprehensive Perl Archive Network)

## `cpanm --installdeps .`

The command `cpanm --installdeps .` is used to install dependencies for a Perl project located in the current directory (`.`).

**cpanm:** This command is the executable for `cpanminus`, a tool used for installing Perl modules from CPAN (Comprehensive Perl Archive Network) and managing Perl dependencies.

**--installdeps:** This is an option or flag passed to `cpanm` to indicate that it should install dependencies. Dependencies are other Perl modules that are required by the project in order for it to function properly.

.: This is a special symbol representing the current directory. In this context, it indicates that `cpanm` should look for a `cpanfile` or `META.json` file in the current directory and install the dependencies listed therein.

When we run `cpanm --installdeps .`, `cpanm` scans the current directory for a `cpanfile` or `META.json` file, which typically specifies the dependencies required by the Perl project. It then installs all of the dependencies listed in that file, ensuring that the project has all of the necessary modules to run successfully.

```
sudo cpan install Switch JSON LWP::UserAgent Config::Simple
```

The command is used to install Perl modules from CPAN (Comprehensive Perl Archive Network) with elevated privileges.

**sudo:** This command is used to execute another command with superuser privileges. This has been explained in earlier sections.

**cpan:** This command is the command-line interface for interacting with CPAN, the Perl module repository. It's used to install, manage, and configure Perl modules.

**install:** This is a sub-command of `cpan` that specifies the action to be performed, which in this case is the installation of one or more Perl modules.

**Switch, JSON, LWP::UserAgent, Config::Simple:** These are the names of the Perl modules that we want to install.

The `::` symbol is used to denote namespaces and package separators. When used in module names, it indicates that the module is part of a particular namespace or package hierarchy.

**Switch:** This is the name of a Perl module. It's a core module in older versions of Perl that provides a way to implement the switch statement, which is similar to a series of if-elsif-else statements but with a more concise syntax.

**JSON:** This is another Perl module used for parsing and generating JSON (JavaScript Object Notation) data. It provides functions and methods for encoding Perl data structures into JSON format and decoding JSON data into Perl data structures.

**LWP::UserAgent:** This module is part of the LWP (Library for WWW in Perl) distribution. It provides an object-oriented interface for making HTTP requests and handling responses. The :: in LWP::UserAgent indicates that UserAgent is a sub-module or class within the LWP namespace.

**Config::Simple:** Similar to LWP::UserAgent, Config::Simple is also a Perl module. It provides a simple way to read and write configuration files in a human-readable format. The :: in Config::Simple indicates that Simple is a sub-module or class within the Config namespace.

In summary, when we run **sudo cpan install Switch JSON LWP::UserAgent Config::Simple**, we are instructing the system to install the specified Perl modules (Switch, JSON, LWP::UserAgent, Config::Simple) using the cpan command. The sudo command ensures that the installation process has the necessary administrative privileges to modify system files and install the modules. Once the installation is complete, we will have access to the functionality provided by these Perl modules in the Perl scripts and applications.

**sudo perl nipe.pl install**

**Sudo:** As explained above, provides user superuser privileges

**perl:** This command is the Perl interpreter, used to execute Perl scripts. In this case, it's used to run the Perl script named nipe.pl.

**nipe.pl:** This is the name of the Perl script that you want to execute. It's assumed to be located in the current directory, as no path is specified.

**install:** This is an argument or parameter passed to the Perl script nipe.pl. In this context, it likely indicates that the script should perform an installation process.

In conclusion, the commands stated in 1.4.1 should install the required code and applications for running nipe.pl

## 2.1.6) Checking anonymous status

If `locate nipe.pl` yielded an output (`nipe.pl` exists), then the script will check on its anonymous status. This will be explained in this section. The following commands were used to find out the anonymous status of the device.

```
nipelocation=$(locate nipe.pl | head -n1)
```

**locate nipe.pl:** This part of the command uses the `locate` command to search for all occurrences of the file named `nipe.pl` on the system. `locate` searches a pre-built database of filenames and their paths, making it faster than searching the entire filesystem.

`|`: This symbol is a pipe operator used to pass the output of one command as input to another command.

**head -n1:** This part of the command takes the output of `locate` and passes it to the `head` command with the `-n1` option. The `-n1` option tells `head` to output only the first line of its input.

So, when we run `locate nipe.pl | head -n1`, the system searches for all occurrences of `nipe.pl`, and `head -n1` then selects the first occurrence from the list. Finally, the location of the first occurrence is stored in the variable `nipelocation` using the `=$(locate nipe.pl | head -n1)` command.

```
nipefolder=$(echo $nipelocation | sed 's/\/nipe.pl//g')
```

**echo \$nipelocation:** This part of the command uses the `echo` command to print the value stored in the variable `nipelocation`, which is the location of the `nipe.pl` file.

`|`: This is the pipe operator, which is used to pass the output of one command as input to another command.

**sed 's/\/nipe.pl//g':** This part of the command uses the `sed` command to perform a search and replace operation on the input text. Specifically, it searches for the substring `/nipe.pl` and replaces it with an empty string (`"`). The `g` flag at the end indicates that this operation should be applied globally, meaning it will replace all occurrences of `/nipe.pl` in

the input text. Hence, this removes the /nipe.pl part from the path, leaving only the folder containing the script.

**nipefolder=\$(...):** This part of the command captures the output of the entire command sequence (echo \$nipelocation | sed 's/\nipe.pl//g') and assigns it to the variable nipefolder.

**cd \$nipefolder**

**cd:** This is the command used to change directories.

**\$nipefolder:** This is a variable containing the path of the directory you want to change to. When \$nipefolder is expanded, it represents the directory path stored in the variable.

When we run cd \$nipefolder, it changes the current working directory to the directory stored in the variable \$nipefolder, allowing us to navigate to the directory containing the nipe.pl file. This is essential when we want to execute commands or perform operations within the directory where the nipe.pl script is located.

**sudo perl nipe.pl start**

**sudo:** As explained above, provides superuser privileges

**perl:** This command is the Perl interpreter, used to execute Perl scripts.

**nipe.pl:** This is the name of the Perl script that we want to execute. It is assumed to be located in the current directory, which has been completed in the previous step.

**start:** This is an argument or parameter passed to the nipe.pl script. The specific behavior of the script depends on how it's programmed to handle this argument. In this context, it indicates that the script should initiate a process or action related to network anonymization.

**sudo perl nipe.pl status**

sudo: Explained previously

perl: Explained previously

nipe.pl: Explained previously

**status:** This is an argument or parameter passed to the nipe.pl script. In this context, it indicates that the script should provide information about the current status or configuration of the network anonymization process.

```
nipetrue=$(sudo perl nipe.pl status | grep Status | awk '{print $3}')  
#variable for grepping for the word "true"
```

**sudo perl nipe.pl status:** Explained in previous section

**|:** Pipe operator

**grep Status:** This part of the command filters the output to only include lines containing the word "Status".

**awk '{print \$3}':** This part of the command uses the awk tool to extract the third column of text from each line of input. In this context, it is extracting the value of the status, assuming it's in the third column.

**nipetrue=\$(...):** This part of the command assigns the output of the preceding command sequence to the variable nipetrue.

Hence when we run `nipetrue=$(sudo perl nipe.pl status | grep Status | awk '{print $3}')`, the system executes the nipe.pl script with elevated privileges, retrieves its status, filters it to include only lines containing the word "Status", extracts the value of the status (assumed to be in the third column), and stores it in the variable nipetrue. This allows subsequent parts of the script to make decisions based on whether the status is "true" or not.

```
if [ $nipetrue == 'true' ]
```

**if [ \$nipetrue == 'true' ]:** This line initiates an if statement in Bash. It checks whether the value stored in the variable nipetrue is equal to the string "true". If the value stored in nipetrue is 'true' it leads to the **'then'** script pathway, otherwise, it will lead to the **'else'** script pathway.

### **'then' pathway**

**then:** This keyword indicates the start of the code block to be executed if the condition specified in the if statement is true.

```
anonIP=$(sudo perl nipe.pl status | grep Ip |  
awk '{print $3}')
```

This command retrieves the anonymous IP address by running `sudo perl nipe.pl status` to get the status of the network configuration.

The output is piped to `grep Ip` to filter out the line containing the IP address information.

Finally, `awk '{print $3}'` extracts the third column, which contains the IP address, and stores it in the variable `anonIP`.

```
echo "$anonIP is the anonymous IP  
address"
```

This command prints a message to the console indicating the anonymous IP address obtained in the previous step.

```
anonIPcountry=$(geoplookup $anonIP | head -n1 | awk  
'{print $5,$6,$7}')
```

This command retrieves the country associated with the anonymous IP address. It uses `geoplookup $anonIP` to perform a lookup of the IP address and retrieve information about its geographical location.

The output is piped to `head -n1` to limit the output to the first line, which typically contains the most relevant information.

Finally, `awk '{print $5,$6,$7}'` extracts the fifth, sixth, and seventh columns, which usually contain the country name, and stores it in the variable `anonIPcountry`. The

additional columns were extracted in order to cater to countries with more than one word (for example, United States).

```
echo "$anonIPcountry is the spoofed  
country name"
```

This command prints a message to the console indicating the country associated with the anonymous IP address, which is often referred to as the "spoofed country name."

**'Else' pathway**

```
echo "You are not anonymous! Please  
ensure anonymousness Exiting now!"  
#string informs user of exiting as not  
anonymous
```

This command uses echo to print a message to the terminal, alerting the user that the network is not configured for anonymity. The message informs the user of the situation and advises them to ensure anonymity.

**exit**

This command terminates the script immediately after printing the message. When the script reaches this point, it halts its execution and exits.

This is particularly relevant in this context because the script has determined that the network is not anonymous, so there's no further action to take regarding network anonymity. Thus, the script exits to prevent any further commands from executing.



2.1.7) Allow the user to specify the address/URL to whois from remote server; save into a variable

```
echo 'Please specify the <requesting  
information from user>'
```

Echo: The shell script prints the text. Informing the user about what action is required.

```
read <variable name>
```

Read: the input given by the user is stored as a variable name given in <variable name>

The above process in chapter 1.5 is repeated for

- The url/address to be nmaped/whois-ed -> stored as variable name url
- The ip address of the remote host -> stored as variable name remotehost
- The user of the remote host -> stored as variable name remoteuser
- The password of the user of the remote host -> stored as variable name remotepass

### 3) Automatically Scan the Remote Server for open ports.

#### 3.1.1) Connect to the Remote Server via SSH

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip"  
echo "Connected to server!"
```

**sshpass:** This is a command-line utility that provides password input for SSH connections. It's particularly useful in scripts where providing a password interactively isn't feasible.

**-p "\$remotepass":** This option specifies the password to be used for the SSH connection. The variable \$remotepass holds the password required to authenticate to the remote server.

**ssh:** This is the command-line tool used to connect to remote systems via the SSH protocol.

**"\$remoteuser@\$remoteip":** This specifies the username (\$remoteuser) and the IP address (\$remoteip) of the remote server to connect to. The variable \$remoteuser holds the username required to log in to the remote server.

**echo "Connected to server!":** This command is executed on the remote server after the SSH connection is established. It simply prints the message "Connected to server!" to the terminal on the remote server.

When we run this command **sshpass -p "\$remotepass" ssh**

**"\$remoteuser@\$remoteip" echo "Connected to server!"**, it attempts to establish an SSH connection to the remote server using the provided username, password, and IP address. If the connection is successful, it executes the echo command on the remote server to print the message "Connected to server!" to the terminal.

### 3.1.2) Display the details of the remote server

```
remoteipcountry=$(sudo geoiplookup $remoteip  
| head -n1 | awk '{print $5,$6,$7}'
```

**sudo:** Explained previously, tasked to provide super user permissions

**geoiplookup \$remoteip** is used to perform a geographic lookup of an IP address (\$remoteip). It typically provides information about the geographical location associated with that IP address, such as the country, city, and sometimes more detailed location data.

**head:** This part of the command takes the first 10 output of sudo geoiplookup \$remoteip.

**-n1:** Selects only the first line of output

**awk '{print \$5,\$6,\$7}':** This part of the command uses awk to process the output from head -n1. It selects the 5th, 6th, and 7th columns of text from the input (which typically contain the country information) and prints them. The additional columns were extracted in order to cater to countries with more than one word (for example, United States).

**remoteipcountry=\$(...):** This part of the command captures the processed country information and stores it in the variable \$remoteipcountry for later use.

```
echo "The country of the remote server is  
$remoteipcountry"
```

Informs the user of the remoteipcountry in a string sentence by using “\$” to call out the variable.

```
echo "The IP of the remote server is $remoteip"
```

Informs the user of the remoteip in a string sentence by using “\$” to call out the variable.

```
timeup=$(sudo uptime | awk '{print $1}')
```

**timeup=**: This part initiates the assignment of a value to the variable named timeup. Variables in shell scripts are assigned values using this syntax.

**\$(...)**: This is command substitution. It allows the output of a command to replace the command itself. Whatever output is produced by the command inside the parentheses will be assigned to the variable timeup.

**sudo uptime**: This command is executed with elevated privileges using sudo. uptime is a command that provides information about how long the system has been running, along with other system-related information. .

**|**: This is the pipe operator, used to redirect the output of one command (in this case, sudo uptime) as input to another command (in this case, awk).

**awk '{print \$1}'**: awk is a text-processing tool used for data extraction and reporting. In this command:

**'...'** encloses the actual awk script.

**{print \$1}** is an awk command that instructs awk to print the first column of each input line. Columns are separated by whitespace by default. \$1 refers to the first column.

```
echo "This server has been up for $timeup"
```

```
echo "HH:MM:SS"
```

This command informs the user of the amount of hours the server has been up, while displaying the format of the respective time below.

```
timeupusers=$(sudo uptime | awk '{print $(NF-6)}')
```

**timeupusers=**: This starts the assignment of a value to the variable named timeupusers.

**\$(...)**: Command substitution is used here, allowing the output of a command to replace the command itself. Whatever output is produced by the command inside the parentheses will be assigned to the variable timeupusers.

**sudo uptime**: This command is executed with elevated privileges using sudo. uptime is a command that displays system uptime and current time, along with other information like the number of users logged in.

**|**: The pipe operator is used to redirect the output of sudo uptime as input to the next command in the pipeline, awk.

**awk '{print \$(NF-6)}'**: This is an awk command that prints the value of the field which is six fields before the last field (NF refers to the total number of fields in a record).

**\$(NF-6)**: This expression evaluates to the value of the field which is six fields before the last field. In the context of sudo uptime, this field contains the number of users currently logged in.

Overall, the command fetches the output of sudo uptime, extracts the number of users currently logged in using awk, and assigns it to the variable timeupusers. Therefore, timeupusers would contain the number of users currently logged in.

**echo "There are \$timeupusers user(s) currently logged in to this server"**

This command informs the user of the number of users currently logged into the server

4.1) Scan (NMAP and whois) the domain/url provided by the user on the remote server and save the results of the scan onto the remote server(Ubuntu)

4.1.1) Changing permissions of /var and /var/log folder

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip"  
"sudo -S chmod 777 /var /var/log"
```

**sshpass -p "\$remotepass"**: This part of the command uses sshpass to provide the password (\$remotepass) for the SSH connection non-interactively. sshpass is a utility that allows providing passwords automatically to SSH connections.

**-p**: Indicates where you will provide the password (in this case the variable \$remotepass)

**ssh "\$remoteuser@\$remoteip"**: This connects to the remote server using SSH. \$remoteuser is the username used to log in to the remote server, and \$remoteip is the IP address of the remote server.

**"sudo -S chmod 777 /var /var/log"**: Once the SSH connection is established, this part of the command executes the specified command on the remote server. Here, sudo is used to run the chmod command with elevated privileges (-S flag is used to read the password from standard input). chmod 777 /var /var/log changes the permissions of the /var and /var/log directories to allow read, write, and execute permissions for all users.

4.1.2) Create a scan log folder to document scan details

```
sshpass -p "$remotepass" ssh  
"$remoteuser@"@"$remoteip" 'sudo -S touch
```

**/var/log/scanned.log && sudo -S chmod 777**

**/var/log/scanned.log'**

**sshpass -p "\$remotepass" ssh "\$remoteuser"@"\$remoteip":** Explanation is as of previous command

**sudo -S touch /var/log/scanned.log:** This part of the command does the following:

**sudo:** Executes the subsequent command with elevated privileges.

**-S:** Instructs sudo to read the password from standard input.

**touch /var/log/scanned.log:** The touch command is used to create a new file named scanned.log in the /var/log directory. If the file already exists, touch updates its timestamp to the current time. The file path /var/log/scanned.log specifies the location where the file should be created.

**&&:** This is a logical operator that represents the "and" condition. It ensures that the command following it is executed only if the previous command (before &&) succeeds

**.sudo -S chmod 777 /var/log/scanned.log:** This part of the command does the following:

**sudo:** Executes the subsequent command with elevated privileges.

**-S:** Instructs sudo to read the password from standard input.

**chmod 777 /var/log/scanned.log:** The chmod command is used to change the permissions of the file scanned.log located in the /var/log directory.

Chmod 777 means:

7 (read, write, and execute) for the owner of the file.

7 (read, write, and execute) for the group associated with the file.

7 (read, write, and execute) for other users (everyone else).

**/var/log/scanned.log:** Specifies the path to the file whose permissions are being modified.

#### 4.1.3) whois/nmap scanning and saving into /var/log

```
echo "Now checking the whois data of the given  
URL/Address, saving scanned whois data to  
/var/log as scanWHOIS.txt"
```

Information string informing user that the scan is starting

```
sshpass -p "$remotepass" ssh  
"$remoteuser@$remoteip" "sudo -S whois $url  
> /var/log/scanWHOIS.txt "
```

**sshpass -p "\$remotepass" ssh "\$remoteuser@\$remoteip":** Explained in previous sections

**"sudo -S whois \$url > /var/log/scanWHOIS.txt "**

**Sudo -S:** explained in previous sections

**whois \$url:** inside this ssh session, the 'whois' command is executed with elevated privileges.

**>:** indicates the file is redirected

**/var/log/scanWHOIS.txt:** file is redirected to a file named "scanWHOIS.txt" in the pathway /var/log. This creates a file named scanWHOIS.txt if the file is not present.

**This process is thereafter repeated for the nmap command by substituting the whois command with nmap, and gives the output into scanNMAP.txt**

#### 4.1.4) Saving scanned details into scanned.log



## **whoisTIME=\$(date +%H:%M:%S)**

**whoisTIME=**: This segment begins the assignment of a value to the variable `whoisTIME`. In shell scripting, variables are declared by specifying the variable name followed by an equals sign (=).

**\$(...)**: Command substitution is used here to capture the output of the command within the parentheses.

**date**: Invokes the date command.

**+%H:%M:%S**: This specifies the format in which the date and time should be displayed. Specifically:

**%H**: Represents the hour (00-23) in 24-hour format.

**%M**: Represents the minute (00-59).

**%S**: Represents the second (00-59).

So, `date +%H:%M:%S` formats the output to display the current time in hours:minutes:seconds.

Putting it all together, `whoisTIME=$(date +%H:%M:%S)` captures the current time in hours:minutes:seconds format and assigns it to the variable `whoisTIME` for later use in the script.

## **whoisDATE=\$(date +%Y-%m-%d)**

**whoisDATE=**: This segment begins the assignment of a value to the variable `whoisDATE`. In shell scripting, variables are declared by specifying the variable name followed by an equals sign (=).

**\$(...)**: Command substitution is used here to capture the output of the command within the parentheses.

**date +%Y-%m-%d**: This command invokes the date utility, which displays or sets the system date and time. Here's the breakdown of its components:

**date**: Invokes the date command.

**+%Y-%m-%d**: This specifies the format in which the date should be displayed. Specifically:

**%Y**: Represents the year (e.g., 2023, 2024, etc.).

**%m**: Represents the month (01-12).

**%d**: Represents the day of the month (01-31).

Therefore, `whoisDATE=$(date +%Y-%m-%d)` formats the output to display the current date in the year-month-day format into the variable “`whoisDATE`” for later use in the script.

**whoisDAY=\$(date +"%A")**

**whoisDAY=**: This segment begins the assignment of a value to the variable whoisDAY. In shell scripting, variables are declared by specifying the variable name followed by an equals sign (=).

**\$(...)**: Command substitution is used here to capture the output of the command within the parentheses.

**date +"%A"**: This command invokes the date utility, which displays or sets the system date and time. Here's the breakdown of its components:

**date**: Invokes the date command.

**+"%A"**: This specifies the format in which the day of the week should be displayed.

**%A**: Represents the full name of the day of the week (e.g., Monday, Tuesday, etc.).

Therefore, **whoisDAY=\$(date +"%A")** formats the output to display the current day of the week as its full name.

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "echo "A whois  
scan was completed on $whoisDAY, $whoisDATE at $whoisTIME on $url"  
>> /var/log/scanned.log"
```

**sshpass -p "\$remotepass" ssh "\$remoteuser@\$remoteip"**: This was explained in previous chapters.

**"echo "A whois scan was completed on \$whoisDAY, \$whoisDATE at \$whoisTIME on \$url"**: This echoes the sentence with dynamic variables including the scan date, time, day and the targeted ip of the scan.

**>> /var/log/scanned.log"**

**>>**: This appends the echoed sentence into the file "scanned.log" in the /var/log folder of the remote server.

**This process was thereafter repeated for the nmap command by substituting the whois command with nmap, and gives the output into scanNMAP.txt**

**The variables \$nmapDATE, \$nmapTIME, \$nmapDAY were given for the NMAP scan in place of the whois scan variables respectively.**

#### 4.1.5) Retrieve the results of the scan from your local machine

The following commands were used

```
#Obtaining file via ftp
# Connect to FTP server and download scanNMAP.txt and scanWHOIS.txt
ftp -n $remoteip <<EOF
quote USER $remoteuser
quote PASS $remotepass
cd /var/log
lcd ~
get scanNMAP.txt
get scanWHOIS.txt
quit
EOF
```

This script uses the FTP (File Transfer Protocol) command-line client to perform a series of FTP operations on a remote server.

### **ftp -n \$remoteip <<EOF**

**ftp**: Invokes the FTP command-line client.

**-n**: This flag instructs FTP not to attempt to auto-login upon initial connection.

**\$remoteip**: The IP address of the remote server.

**<<EOF**: This starts a here document, which allows for inline input of multiple lines until a specified delimiter (EOF in this case). The commands following <<EOF until EOF are fed into the standard input of the ftp command.

**quote USER \$remoteuser**: This sends a command to the FTP server to authenticate the user with the username stored in the variable \$remoteuser.

**quote**: This FTP command sends arbitrary commands to the server.

**USER**: This command is used to send the username to the FTP server.

**\$remoteuser**: The username used for authentication, stored in a variable.

**quote PASS \$remotepass**: This sends a command to the FTP server to authenticate the user with the password stored in the variable \$remotepass.

**PASS**: This command is used to send the password to the FTP server.

**\$remotepass**: The password used for authentication, stored in a variable.

**cd /var/log**: This changes the working directory on the FTP server to /var/log.

**cd**: This FTP command is similar to the shell command and is used to change the directory on the server. In this case, it changes the directory to /var/log

**lcd**: lcd stands for "local change directory." It is used within the context of an FTP session to change the local directory on your machine, where the files will be downloaded to. In this case, ~ means the user's home directory.

**get scanNMAP.txt**: This retrieves the file named scanNMAP.txt from the current directory on the FTP server and downloads it to the local machine.

**get scanWHOIS.txt:** This retrieves the file named scanWHOIS.txt from the current directory on the FTP server and downloads it to the local machine

**quit:** This FTP command is used to terminate the FTP session and exit the FTP command-line client.

**EOF:** This marks the end of the here document, specifying the end of the input fed into the ftp command.

#### 4.1.6) Capturing of traffic using tcpdump and wireshark

```
sudo tcpdump -i eth0 -vv -w capture.pcap
```

**sudo:** This command is used to execute the subsequent command (tcpdump) with superuser (root) privileges.

**tcpdump:** This is the command-line packet analyzer tool used to capture and analyze network traffic.

**-i eth0:** This option specifies the network interface to capture packets from. In this case, eth0 is the network interface. You can replace eth0 with the name of any available network interface on your system.

**-vv:** These are verbose options. They instruct tcpdump to display more detailed information about the captured packets. Using multiple -v flags increases the verbosity level. Here, -vv indicates a high level of verbosity.

**-w capture.pcap:** This option specifies the filename to save the captured packets to. In this case, the filename is capture.pcap, and it will be saved in the current working directory. The file format is .pcap, which is a common format used for packet capture files.

### 3) Discussion

In this chapter, we will evaluate the findings that were described in earlier sections. The methodology, results and process will be evaluated to support the recommendations that will be made in the following chapter.

### 3.1) Installations and anonymity check

Executing sudo apt-get update

**Script:**

```
#!/bin/bash

#Installations and Anonymity Check

#Ensuring system is updated to the latest
echo 'Ensuring system is updated to the latest version by executing "'
echo "$(tput blink)Please wait, patience is a virtue$(tput sgr0)" #B
sudo apt-get update #Ensures that packages in system are up to date
```

**Output:**

```
(kali@kali)-[~]
$ bash networkp.sh
Ensuring system is updated to the latest version by executing "sudo apt-get update"
Please wait, patience is a virtue
[sudo] password for kali:
Get:1 http://archive-4.kali.org/kali kali-rolling InRelease [41.5 kB]
Get:2 http://archive-4.kali.org/kali kali-rolling/main amd64 Packages [19.9 MB]
16% [2 Packages 132 kB/19.9 MB 1%] 28.8 kB/s 38min 58s

(kali@kali)-[~]
$ bash networkp.sh
Ensuring system is updated to the latest version by executing "sudo apt-get update"
Please wait, patience is a virtue
[sudo] password for kali:
Get:1 http://archive-4.kali.org/kali kali-rolling InRelease [41.5 kB]
Get:2 http://archive-4.kali.org/kali kali-rolling/main amd64 Packages [19.9 MB]
Get:3 http://archive-4.kali.org/kali kali-rolling/main amd64 Contents (deb) [46.5 MB]
Get:4 http://archive-4.kali.org/kali kali-rolling/contrib amd64 Packages [115 kB]
Get:5 http://archive-4.kali.org/kali kali-rolling/non-free amd64 Packages [193 kB]
Get:6 http://archive-4.kali.org/kali kali-rolling/non-free amd64 Contents (deb) [864 kB]
Get:7 http://archive-4.kali.org/kali kali-rolling/non-free-firmware amd64 Packages [33.1 kB]
Fetched 67.6 MB in 1min 35s (715 kB/s)
Reading package lists... Done
```

Prior to checking the installed programs, sudo apt-get update was used to ensure that our system's package database is up-to-date. This command fetches the latest information about available packages from the repositories configured on your system.

## Checking for geolookup and sshpass application

### Script

```
function checkgeolookup() #creates function geolookup
{
    if [ $(command -v geolookup) == -z ]      #if geolookup is in the commands database,
    then
        echo "geolookup is required but it's not installed. Installing geolookup now."
        sudo apt-get install geolookup #installs geolookup
    else
        echo "geolookup application is installed" #string informing user geolookup already installed
    fi
}
checkgeolookup #Calls out geolookup function

function checksshpass() #creates function checksshpass
{
    if [ $(command -v sshpass) == -z ]      #if sshpass is in the commands database, output
    then
        echo "sshpass is required but it's not installed. Installing sshpass now." #S
        sudo apt-get install sshpass #installs sshpass
    else
        echo "sshpass application is installed" #string informing user sshpass already installed
    fi
}
checksshpass #Calls out checksshpass function
```

### Output

```
geolookup application is installed
sshpass application is installed
```

The command `-v <command name> command` is used in Unix-like operating systems to determine the path or location of a specified command within the system's executable search path.

When the commands are used individually, it shows an output for both geolookup and sshpass

```
(kali@kali)-[~]
$ command -v geolookup
/usr/bin/geolookup

(kali@kali)-[~]
$ command -v sshpass
/usr/bin/sshpass
```

In the case of searching for the application or command named "application123" exists it gives no output. Following which will proceed to the installation of geolookup and/or ssh pass.

```
(kali@kali)-[~]
$ command -v application123

(kali@kali)-[~]
$
```

## 3.2) Checking for nipe.pl

As nipe.pl is a perl script, the previously mentioned method to find out if a command exists, will not work. As nipe.pl is a perl script, we used locate to find the location of nipe.pl. In a scenario whereby nipe.pl is not installed, the command locate nipe.pl will not give an output similar to the command -v application123 example mentioned above.

Script

```
function checknipe() #creates function checknipe and checks anonymous status
{
    # Locating nipe.pl and its relevant files
    if [ $(locate nipe.pl) == -z ] #locates for nipe.pl files. If nil found, to install nipe and relevant files
    then
        echo "Unable to locate nipe.pl. Installing nipe.pl now"
        git clone https://github.com/htrgouvea/nipe && cd nipe
        sudo apt-get install cpanminus
        cpanm --installdeps .
        sudo cpan install Switch JSON LWP::UserAgent Config::Simple
        sudo perl nipe.pl install
    else
        #nipe.pl is installed, will check if network is anonymous

```

If however, an output is given like this example below,

```
(kali@kali)-[~]
$ locate nipe.pl
/home/kali/nipe/nipe.pl
```

The script will check if the status is anonymous

```
else
    #nipe.pl is installed, will check if network is anonymous
    echo "nipe.pl is installed, checking if network is anonymous"
    nipelocation=$(locate nipe.pl | head -n1)
    nipefolder=$(echo $nipelocation | sed 's/\/nipe.pl//g')
    cd $nipefolder
    sudo perl nipe.pl start
    sudo perl nipe.pl status
    nipetrue=$(sudo perl nipe.pl status | grep Status | awk '{print $3}')
    if [ $nipetrue == 'true' ]
    then
        anonIP=$(sudo perl nipe.pl status | grep Ip | awk '{print $3}')
        echo "$anonIP is the anonymous IP address"
        anonIPcountry=$(geoipllookup $anonIP | head -n1 | awk '{print $5,$6,$7}')
        echo "$anonIPcountry is the spoofed country name"
    else
        echo "You are not anonymous! Please ensure anonymousness Exiting now!"
        exit
    fi
fi
```

Output



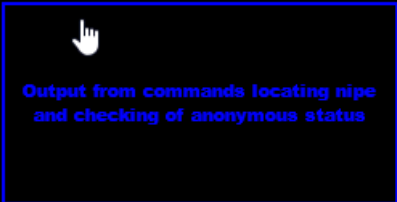
Upon execution of the command, if nipe.pl is installed it gives the following output:

```
(kali@kali)-[~]
$ bash networkp.sh
Ensuring system is updated to the latest version by executing "sudo apt-get update"

[sudo] password for kali:
Hit:1 http://http.kali.org/kali kali-rolling InRelease
Reading package lists... Done
geoplookup application is installed
sshpas application is installed
nipe.pl is installed, checking if network is anonymous

[+] Status: true
[+] Ip: 185.220.101.83

185.220.101.83 is the anonymous IP address
Germany is the spoofed country name
```



Using the above example, the pathway of nipe.pl is in /home/kali/nipe/nipe.pl. Nipe.pl's folder is obtained using the \$nipefolder variable command which removes the "/nipe.pl" of the pathway directory of nipe. This gives the variable as /kali/home/nipe. The directory is then changed to that variable so that the perl nipe.pl commands can be executed. This is illustrated below:

```
else #nipe.pl is installed, will check if network is anonymous
echo "nipe.pl is installed, checking if network is anonymous"
nipelocation=/kali/home/nipe/nipe.pl #Obtains the nipe location
nipefolder=/kali/home/nipe (nipe.pl//g') #Use nipe location to get
cd $nipefolder changes user's directory to /kali/home/nipe (where nipe.pl is located) #Starting nipe.pl
sudo perl nipe.pl start
sudo perl nipe.pl status
nipe=true$(sudo perl nipe.pl status | grep Status | awk '{print $3}') #vari:
if [ $nipe == 'true' ] #If managed to gr
then
anonIP=$(sudo perl nipe.pl status | grep Ip | awk '{print $3}')
echo "$anonIP is the anonymous IP address" #String message indic
anonIPcountry=$(geoplookup $anonIP | head -n1 | awk '{print $5,$6,$7}')
echo "$anonIPcountry is the spoofed country name" #string informing the
else
echo "You are not anonymous! Please ensure anonymousness Exiting now!" #s
exit
fi
```

Following that, we execute sudo perl nipe.pl start and sudo perl nipe.pl status and grep for the word "Status" and awk for the third column's word and check if it is the word "true". Awk for the third column is denoted by the "\$3". Below is an illustration.



```
(kali@kali)-[~]
$ bash networkp.sh
Ensuring system is updated to the latest version by executing "sudo apt-get update"

[sudo] password for kali:
Hit:1 http://http.kali.org/kali kali-rolling InRelease
Reading package lists... Done
geoipllookup application is installed
sshsu application is installed
nipe.pl is installed, checking if network is anonymous
$1 $2 $3
[+] Status: true This line is grepped with "grep Status" command
[+] Ip: 185.220.101.83

185.220.101.83 is the anonymous IP address
Germany is the spoofed country name
```

Output from commands locating nipe and checking of anonymous status

If the variable for \$nipetrue is not “true”, the script will inform the user that the session is not anonymous and exit the session.

Nipe.pl is a script which is designed to route all traffic through the Tor network in order to maintain anonymity. This allows users to browse the internet anonymously and route their internet traffic through a series of nodes, obscuring the user’s IP address and location. This is particularly important when running an FTP.

Anonymity helps protect our identity and sensitive information from being exposed to the server. Also, by hiding our real IP address, we minimise the risk of unauthorised access to our data, reducing the chances of being targeted by cyberattacks and surveillance.

Depending on the nature of the data being transferred, there may be requirements for the transferring of data to be anonymous and confidential as well.

### 3.3) If anonymous, to display the spoof country name

```
(kali㉿kali)-[~]
└─$ bash networkp.sh
Ensuring system is updated to the latest version by executing "sudo apt-get update"

[sudo] password for kali:
Hit:1 http://http.kali.org/kali kali-rolling InRelease
Reading package lists... Done
geoipllookup application is installed
sshsppass application is installed
nipl.pl is installed, checking if network is anonymous

$1 $2 $3
[+] Ip: 185.220.101.83
185.220.101.83 is the anonymous IP address
Germany is the spoofed country name
```

grep IP obtains this line as output

Output from commands locating nipl and checking of anonymous status

Grep Ip obtains the line stated in the image above.

Awk '{print \$3}' is the command used to obtain the ip address from the grepped line.

Following that, the anonIP=\$(`<stated commands>`) will store the ip address under the variable anonIP.

The command geoipllookup is then used to give information about the geological address of the IP. Below is an example of how the output looks on its own

```
(kali㉿kali)-[~]
└─$ geoipllookup 185.220.101.83
GeoIP Country Edition: DE, Germany
```

The command head -n1 is used to give the first result displayed. Following that the awk '{print \$5,\$6,\$7}' is used to print the countries of the spoofed address. In this case, Germany is in the fifth column which is denoted by \$5. However additional columns are added in case of country names that have more than one word. For example, The United States.

Using the command anonIPcountry=\$(`<commands used above>`), the spoof's country is stored as a variable name anonIPcountry.

3.4) Allow the user to specify the address/URL to whois from remote server; save into a variable.

Script:

```
#obtains the url to be scanned, remote user and password an remote host ip
echo 'Please specify the URL/address to scan from the remote server'
read url #read user input to save as a variable $url
echo "Please specify the ip of the remote host" #string asking for ip of remote host
read remoteip #reads user input as a variable $remoteip
echo "Please specify the user of the remote host" #string asking for user of the remote host
read remoteuser #reads user input as variable $remoteuser
echo "Please type the password of the user in the previous question" #string asking user for password of the remote user
read remotepass #reads user input as variable $remotepass
sleep 1 #spaces out information to prevent user information overload
```

Output

```
Please specify the URL/address to scan from the remote server
192.168.31.130
Please specify the ip of the remote host
192.168.31.131
Please specify the user of the remote host
tc
Please type the password of the user in the previous question
tc
```

In this case, the script prompts the user to input a series of inputs that will read as the respective variables. It is also important to note that the password was visible when typing out the remote host password. If there were prying eyes around the user while he uses his main machine, this would greatly compromise the security of the remote account used by the user. This could lead to a landslide of cybersecurity issues like data breaches, data theft, data manipulation and unauthorised access.

3.5) Connect to the remote server via ssh

script:

```
#Connect to the remote server
echo "Now attempting to connect to the remote server"
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" echo "Connected to server!"
```

Output:

```
Please specify the URL/address to scan from the remote server
192.168.31.130
Please specify the ip of the remote host
192.168.31.131
Please specify the user of the remote host
tc
Please type the password of the user in the previous question
tc
Now attempting to connect to the remote server
Connected to server!
```

Using the sshpass command, once the user manages to connect into the server, it echos "Connected to the server!" indicating that a connection has been established.

### 3.6) Display the details of the remote server

Script:

```
#Obtaining and giving details of the given address/url
echo "Here are the details of the remote server"      #string informing user of remote server details that are about to be given
sleep 1
remoteipcountry=$(sudo geoiplookup $remoteip | head -n1 | awk '{print $5,$6,$7}') #obtains the remote country ip and saves it
echo "The country of the remote server is $remoteipcountry" #information string
sleep 1
echo "The IP of the remote server is $remoteip"      #information string
timeup=$(sudo uptime | awk '{print $1}')
```

#creates variable of the uptime as \$timeup

timeupusers=\$(sudo uptime | awk '{print \$(NF-6)}')

#creates variable of the number of users on the server as \$timeupusers

echo "This server has been up for \$timeup" #information string

echo "HH:MM:SS" #for showing what the above numbers represent in terms of time

sleep 1

echo "There are \$timeupusers user(s) currently logged in to this server" #information string showing number of users in the server

sleep 1

#rest for information overload

Output:

```
Please specify the URL/address to scan from the remote server
192.168.31.130
Please specify the ip of the remote host
192.168.31.131
Please specify the user of the remote host
tc
Please type the password of the user in the previous question
tc
Now attempting to connect to the remote server
Connected to server!
Here are the details of the remote server
The country of the remote server is Address not found
The IP of the remote server is 192.168.31.131
This server has been up for 05:56:50
HH:MM:SS
There are 1 user(s) currently logged in to this server
[sudo] password for tc: tc
[sudo] password for tc: tc
[sudo] password for tc: tc
Now checking the whois data of the given URL/Address, saving scanned whois data to /var/log as scanWHOIS.txt
[sudo] password for tc: tc
Now checking the nmap data of the given URL/Address, saving scanned whois data to /var/log as scanNMAP.txt
[sudo] password for tc: tc
Connected to 192.168.31.131.
220 (vsFTPD 3.0.5)
```

**details of the remote server including the number of users in the server, the country of the remote server and how long the server has been up**

In the above example, the country of the remote server was not found as the remote server's ip given for this example was an internal IP address.

The server's details included the number of users in the server. This is particularly important as it allow us to do security monitoring. Monitoring the number of users connected to the FTP server can help detect unauthorized access or suspicious activity. Given the poor blinding of the user details and password in the previous's section, it raises the possibility of an unwanted user accessing the remote server.

Knowing how long the server has been up is also vital in cybersecurity. In the event of a security incident or system outage, knowing the uptime of the FTP server can aid in incident response and forensic analysis. It provides context for determining when the incident occurred, how long the server has been affected, and what actions need to be taken to restore service and mitigate risks.

### 3.7) Conduct an nmap and whois scan and save the details into the remote computer

Script:

```
#Changing permissions of /var and /var/log folder so that we can put the saved scanned in to the var log file
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S chmod 777 /var /var/log" #uses sshpass to chan
sleep 1

#Create a scan log folder to document everthing that was scanned
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S touch /var/log/scanned.log && sudo -S chmod 777 /var/log/scan

#WHOIS scanning and saving into /var/log
echo "Now checking the whois data of the given URL/Address, saving scanned whois data to /var/log as scanWHOIS.txt" #informat
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S whois $url > /var/log/scanWHOIS.txt " #uses sshpass to do whois
whoisTIME=$(date +"%H:%M:%S") #creates Variable for time of whois scan
whoisDATE=$(date +"%Y-%m-%d") #creates Variable for date of whois scan
whoisDAY=$(date +"%A") #creates Variable for day of whois scan
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "echo "A whois scan was completed on $whoisDAY, $whoisDATE at $whoisTIME
sleep 1

#NMAP scanning and saving into /var/log
echo "Now checking the nmap data of the given URL/Address, saving scanned whois data to /var/log as scanNMAP.txt" #informat
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S nmap -O $url > /var/log/scanNMAP.txt $url" #uses sshpass to do
nmapTIME=$(date +"%H:%M:%S") #creates Variable for time of nmap scan
nmapDATE=$(date +"%Y-%m-%d") #creates Variable for date of nmap scan
nmapDAY=$(date +"%A") #creates Variable for day of nmap scan
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "echo "An nmap scan was completed on $nmapDAY, $nmapDATE at $nmapTIME on
sleep 1
```

Output:

```
The IP of the remote server is 192.168.31.131
This server has been up for 05:56:50
HH:MM:SS
There are 1 user(s) currently logged in to this server
[sudo] password for tc: tc
[sudo] password for tc: tc
[sudo] password for tc: tc
Now checking the whois data of the given URL/Address, saving scanned whois data to /var/log as scanWHOIS.txt
[sudo] password for tc: tc
Now checking the nmap data of the given URL/Address, saving scanned whois data to /var/log as scanNMAP.txt
[sudo] password for tc: tc
```

This section of the script consists of 4 main parts:

- 1) Changing permission of the /var and /var/log folder.
- 2) Creating a scanned.log file to log details of the scans done
- 3) Scanning and saving the results of the scan into the remote server
- 4) Appending main details of scan into scanned.log file

The following command was used to change the level of permissions to enable to user to add a scanned.log file and save the results of the scan into the remote server:

```
sshpas -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S chmod 777 /var /var/log"
```

By default, the permissions for /var and /var/log are 775

```

tc@server:/var$ cd /var
tc@server:/var$ ls -l
total 48
drwxr-xr-x  2 root root   4096 May 24 00:06 backups
drwxr-xr-x 15 root root   4096 Apr 23 11:49 cache
drwxrwxrwt  2 root root   4096 Aug 10 2023 crash
drwxr-xr-x 44 root root   4096 Apr 30 12:58 lib
drwxrwsr-x  2 root staff  4096 Apr 18 2022 local
lrwxrwxrwx  1 owner group    9 Aug 10 2023 lock -> /run/lock
drwxrwxr-x 10 root syslog 4096 May 23 09:46 log

```

using the ls -l command we are able to see the permissions of the files in the folder. Here we can see the default for log is 775

The first digit (7) represents the permissions for the owner (user) of the file or directory.  
The second digit (7) represents the permissions for the group that owns the file or directory.  
The third digit (5) represents the permissions for others (everyone else).  
This is represented by the drwxrwxr-x on the left of the image.  
From the image we can also see that owner of the file is root and the groupowner is syslog.  
As our user (tc) is not the owner nor in the group of the owner, we have to change the permissions to allow tc (part of everyone else) to write.

The following image by Communications (2018), illustrates this well.

Symbolic	Numeric	Permission
---	0	None
--x	1	Execute
-w-	2	Write
-wx	3	Write + Execute
r--	4	Read
r-x	5	Read + Execute
rw-	6	Read + Write
rwX	7	Read + Write + Execute

### **Point for recommendation**

It was also important to note that in the given script, there was no command dedicated to revert the chmod permissions back to the default permission level of 775.

This is vital as in cybersecurity, setting the appropriate permissions ensures that only authorised users or processes can access sensitive data. Incorrect permissions may allow unauthorised users to read or modify confidential information, leading to data breaches or unauthorised disclosure of sensitive data.

Incorrect permissions can result in accidental or intentional data loss or corruption. Hence in this case, data integrity and availability might be affected.

## **3.8) Creating scanned.log folder and giving chmod 777 permissions for the scanned.log file**

The following command was used for the stated action:

```
sshpass -p "$remotepass" ssh "$remoteuser@"$remoteip" 'sudo -S touch /var/log/scanned.log  
&& sudo -S chmod 777 /var/log/scanned.log'
```

Similar to the previous section, the file has to be given chmod 777 permission in order for the user (tc) to store any form of data in the file.

## **3.9) Scanning and saving the results of the scan into the remote server**

The follow command was used for both whois and nmap scans:

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S whois $url >  
/var/log/scanWHOIS.txt "
```



While the script managed to achieve its objective of scanning and saving the results, evaluating the execution of the script highlighted areas for improvement

### Points for recommendation

- 1) The script was lengthy and there were numerous times where the same section of the command was repeatedly typed. For instance, "sshpass -p "\$remotepass" ssh "\$remoteuser@\$remoteip"" was repeatedly typed while scripting in order to execute actions on the ssh server
- 2) On executing the script, the password for the remote user was repeatedly asked, which greatly reduced the effectiveness of automation.
- 3) Due to the nature of the '>' operator, if a scan is repeated, the new information from the scan will overwrite the information in the "scanWHOIS.txt" or "scanNMAP.txt" file. Furthermore, as a user of the remote server, that has conducted multiple scans, it will be labour intensive for the user to manually change the name of the txt file in order to differentiate the detailed logs conducted on different times, dates and IP addresses.

## 3.10) Appending main details of scan into scanned.log file

The following command was used:

```
whoisTIME=$(date +"%H:%M:%S")
```

Creating log files of scans is important as it provides a detailed record of the scans performed, including the date, time, and results of each scan. In the event of a security incident or breach, log files serve as valuable forensic evidence.

Collect the file from the remote computer via FTP or HTTP or any other unsecure protocols.

Script:

```
#Obtaining file via ftp
# Connect to FTP server and download scanNMAP.txt and scanWHOIS.txt
ftp -n $remoteip <<EOF
quote USER $remoteuser
quote PASS $remotepass
cd /var/log
get scanNMAP.txt
get scanWHOIS.txt
quit
EOF
```

Output:



```
Connected to 192.168.31.131.
220 (vsFTPD 3.0.5)
331 Please specify the password.
230 Login successful.
250 Directory successfully changed.
local: scanNMAP.txt remote: scanNMAP.txt
229 Entering Extended Passive Mode (|||63974|)
150 Opening BINARY mode data connection for scanNMAP.txt (715 bytes).
  715      20.05 MiB/s
226 Transfer complete.
WARNING! 17 bare linefeeds received in ASCII mode.
File may not have transferred correctly.
715 bytes received in 00:00 (1.38 MiB/s)
local: scanWHOIS.txt remote: scanWHOIS.txt
229 Entering Extended Passive Mode (|||17248|)
150 Opening BINARY mode data connection for scanWHOIS.txt (2781 bytes).
 2781      5.52 MiB/s
226 Transfer complete.
WARNING! 68 bare linefeeds received in ASCII mode.
File may not have transferred correctly.
2781 bytes received in 00:00 (2.83 MiB/s)
221 Goodbye.
Scan Complete
```

Evaluation and analysis of the mentioned step is explained in scope 2 below:

## 4) Research of selected protocol (File Transfer Protocol)

File transfer protocol (FTP) helps facilitate the transfer of files between a client and a server over a network. The key features include authentication, file listing , uploading, downloading, renaming and deleting files (Comer and Stevens, 2000).

### How it works

Using information from RFC 959 (IETF, 2020) and BasuMallick (2022), here are the details of how FTP works:

FTP is part of the application layer and is involved in moving files between the remote and local systems. When a download occurs, FTP helps obtain the file from the remote server, verify the downloader's credentials and then transfer the file into the local machine.

### Message exchange process

FTP is dependent on two concurrent TCP connections, the control connection and the data connection. It uses the control connection to transmit control information, which includes instructions, usernames and passwords. It uses the data connection to send the file, started through a different port than the control connection.

- 1) The client starts a TCP control connection with the server
- 2) The client uses this TCP control connection to send control information
- 3) After receiving this, the server starts a data connection with the client
- 4) A single data connection is used to transfer only one file.
- 5) During an FTP session, the client keeps an open connection to the control server for issuing commands and receiving responses
- 6) Multiple data connections can be started using the same control information

Below is a screencap obtained from wireshark while the ftp process was done. Where the TCP ports represent the control process, whereas the FTP ports represent the data connection.

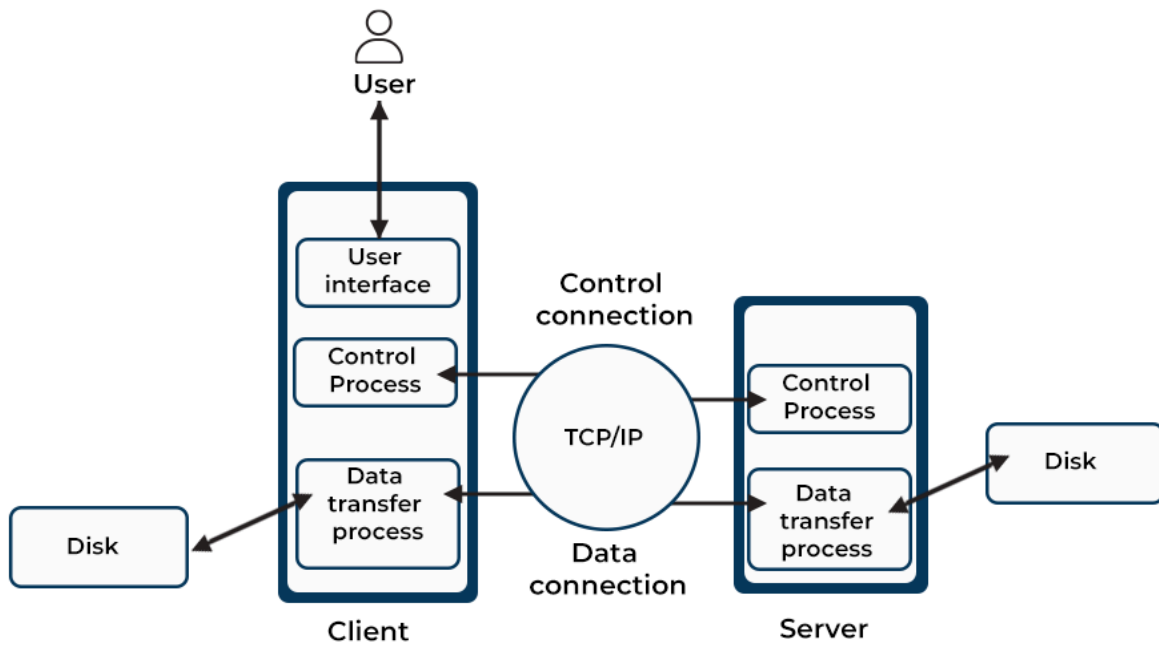
192.168.31.130	192.168.31.131	TCP		74 54076 → 21 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM TSval=1013529452 TSecr=0 WS=2
192.168.31.131	192.168.31.130	TCP	Control	74 21 → 54076 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3938774579 TSecr=1013529452
192.168.31.130	192.168.31.131	TCP		66 54076 → 21 [ACK] Seq=665536 Len=0 TSval=1013529453 TSecr=3938774579
192.168.31.131	192.168.31.130	FTP	Data connection	86 Response: 220 (vsFTPd 3.0.5)
192.168.31.130	192.168.31.131	TCP	Control	66 54076 → 21 [ACK] Seq=665536 Len=0 TSval=1013529453 TSecr=3938774579
192.168.31.130	192.168.31.131	FTP		75 Request: USER tc
192.168.31.131	192.168.31.130	TCP		66 21 → 54076 [ACK] Seq=665536 Len=0 TSval=1013529453 TSecr=3938774579
192.168.31.131	192.168.31.130	FTP		100 Response: 331 Please specify the password.
192.168.31.130	192.168.31.131	FTP		75 Request: PASS tc
192.168.31.131	192.168.31.130	FTP		89 Response: 230 Login successful.
192.168.31.130	192.168.31.131	FTP		80 Request: CWD /var/log
192.168.31.131	192.168.31.130	FTP		103 Response: 250 Directory successfully changed.
192.168.31.130	192.168.31.131	FTP		72 Request: EPSV
192.168.31.131	192.168.31.130	FTP		114 Response: 229 Entering Extended Passive Mode (   24450 )
192.168.31.130	192.168.31.131	FTP		85 Request: RETR scanMAP.txt
192.168.31.131	192.168.31.130	FTP		137 Response: 150 Opening BINARY mode data connection for scanMAP.txt (715 bytes).
192.168.31.130	192.168.31.131	FTP		90 Response: 226 Transfer complete.
192.168.31.131	192.168.31.130	TCP		66 54076 → 21 [ACK] Seq=665536 Ack=403 Win=65138 Len=0 TSval=1013529478 TSecr=3938774604
192.168.31.130	192.168.31.131	FTP		72 Request: EPSV
192.168.31.131	192.168.31.130	FTP		114 Response: 229 Entering Extended Passive Mode (   43842 )
192.168.31.130	192.168.31.131	FTP		86 Request: RETR scanWHOIS.txt
192.168.31.131	192.168.31.130	FTP		139 Response: 150 Opening BINARY mode data connection for scanWHOIS.txt (2781 bytes).
192.168.31.130	192.168.31.131	FTP		90 Response: 226 Transfer complete.
192.168.31.131	192.168.31.130	TCP		66 54076 → 21 [ACK] Seq=665536 Ack=403 Win=65138 Len=0 TSval=1013529478 TSecr=3938774604

tcpstream eq 2032					
No.	Time	Source	Destination	Protocol	Length Info
4458	67.449846	192.168.31.130	192.168.31.131	TCP	74 54076 → 21 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM TSval=1013529452 TSecr=0 WS=2
4459	67.450292	192.168.31.131	192.168.31.130	TCP	74 21 → 54076 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=3938774579 TSecr=1013529452
4460	67.450353	192.168.31.130	192.168.31.131	TCP	66 54076 → 21 [ACK] Seq=665536 Len=0 TSval=1013529453 TSecr=3938774579
4461	67.453487	192.168.31.131	192.168.31.130	FTP	86 Response: 220 (vsFTPd 3.0.5)
4462	67.453532	192.168.31.130	192.168.31.131	TCP	66 54076 → 21 [ACK] Seq=665536 Len=0 TSval=1013529453 TSecr=3938774579
4463	67.453686	192.168.31.130	192.168.31.131	FTP	75 Request: USER tc
4464	67.453863	192.168.31.131	192.168.31.130	TCP	66 21 → 54076 [ACK] Seq=665536 Ack=10 Win=65280 Len=0 TSval=1013529453 TSecr=3938774579
4465	67.454221	192.168.31.131	192.168.31.130	FTP	100 Response: 331 Please specify the password.
4466	67.454403	192.168.31.130	192.168.31.131	FTP	75 Request: PASS tc
4467	67.469548	192.168.31.131	192.168.31.130	FTP	89 Response: 230 Login successful.
4468	67.469834	192.168.31.130	192.168.31.131	FTP	80 Request: CWD /var/log
4469	67.470218	192.168.31.131	192.168.31.130	FTP	103 Response: 250 Directory successfully changed.
4470	67.470401	192.168.31.130	192.168.31.131	FTP	72 Request: EPSV
4471	67.471088	192.168.31.131	192.168.31.130	FTP	114 Response: 229 Entering Extended Passive Mode (   24450 )
4475	67.471815	192.168.31.130	192.168.31.131	FTP	85 Request: RETR scanMAP.txt
4476	67.472281	192.168.31.131	192.168.31.130	FTP	137 Response: 150 Opening BINARY mode data connection for scanMAP.txt (715 bytes).
4484	67.473033	192.168.31.131	192.168.31.130	FTP	90 Response: 226 Transfer complete.
4485	67.473294	192.168.31.130	192.168.31.131	TCP	66 54076 → 21 [ACK] Seq=665536 Ack=258 Win=65282 Len=0 TSval=1013529476 TSecr=3938774602
4486	67.473437	192.168.31.130	192.168.31.131	FTP	72 Request: EPSV
4487	67.473857	192.168.31.131	192.168.31.130	FTP	114 Response: 229 Entering Extended Passive Mode (   43842 )
4491	67.474390	192.168.31.130	192.168.31.131	FTP	86 Request: RETR scanWHOIS.txt
4492	67.474778	192.168.31.131	192.168.31.130	FTP	139 Response: 150 Opening BINARY mode data connection for scanWHOIS.txt (2781 bytes).
4500	67.475339	192.168.31.131	192.168.31.130	FTP	90 Response: 226 Transfer complete.
4501	67.475367	192.168.31.130	192.168.31.131	TCP	66 54076 → 21 [ACK] Seq=665536 Ack=403 Win=65138 Len=0 TSval=1013529478 TSecr=3938774604

Below is a diagram explaining the components involved in the FTP process (BasuMallick, 2022).



## HOW DOES FTP WORK?



# Message Format

Unlike HTTP, FTP does not have a predefined header structure. FTP operates on a command-response model.

An FTP message consists of an ASCII string with 2 parts.

**Commands (from client to server):** Commands are instructions sent by the client to the server to perform specific actions, such as listing directories, uploading files, downloading files, renaming files, etc.

Each command consists of a command code followed by optional parameters.

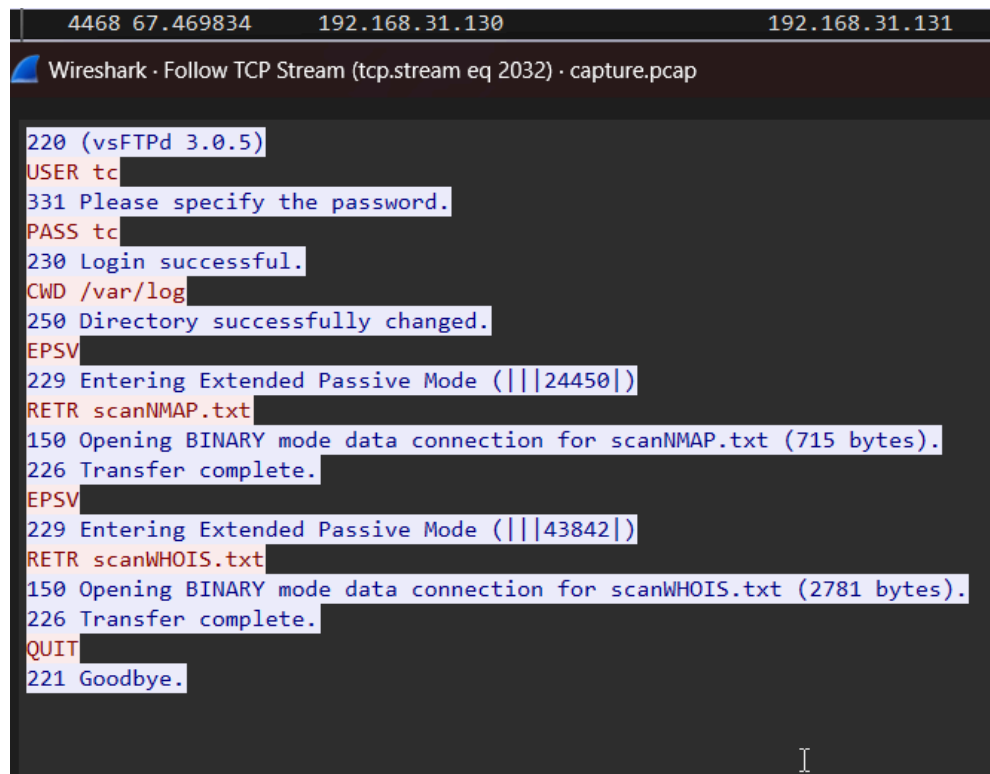
Command codes are typically uppercase alphabetic strings (e.g., "LIST", "RETR", "STOR", "DELE").

**Responses (Server to client):** Feedback messages sent by the server to the client in response to commands.

Each response consists of a three-digit status code followed by an optional message.

The FTP process done in scope one can be used to explain this.

The following screen cap was obtained from the pcap file that captured the traffic flow of the ftp.



```
4468 67.469834 192.168.31.130 192.168.31.131
Wireshark · Follow TCP Stream (tcp.stream eq 2032) · capture.pcap

220 (vsFTPD 3.0.5)
USER tc
331 Please specify the password.
PASS tc
230 Login successful.
CWD /var/log
250 Directory successfully changed.
EPSV
229 Entering Extended Passive Mode (||24450|)
RETR scanNMAP.txt
150 Opening BINARY mode data connection for scanNMAP.txt (715 bytes).
226 Transfer complete.
EPSV
229 Entering Extended Passive Mode (||43842|)
RETR scanWHOIS.txt
150 Opening BINARY mode data connection for scanWHOIS.txt (2781 bytes).
226 Transfer complete.
QUIT
221 Goodbye.
```

The blue highlighted text indicates the message from the server

The red highlighted text indicates the message from the client server.

The following images show the translated information taken from IETF (2020). From IETF (2020), we can see code 220 means the service is ready for a new user, and code 331 indicates the username has been accepted and is awaiting input for the password. Code 230 confirms the log in process.

**220** Service ready for new user.

#### 4.1. FTP COMMANDS

##### 4.1.1. ACCESS CONTROL COMMANDS

The following commands specify access control identifiers (command codes are shown in parentheses).

**USER** NAME (**USER**)

The argument field is a Telnet string identifying the **user**.

**331** User name okay, need password.

**PASSWORD** (**PASS**)

The argument field is a Telnet string specifying the user's **password**. This command must be immediately preceded by the user name command, and, for some sites, completes the user's

**230** User logged in, proceed.

## Flags

**These are commonly used flags that may be used alongside the ftp command to change its behaviour (SolarWinds, 2022).**

**-v** Suppresses verbose display of remote server responses.

By default, FTP displays the responses from the remote server and provides feedback on the status of commands and transfers. By using the -v flag it makes the output cleaner.

**-n** Suppresses auto-login upon initial connection.

Normally, when you initiate an FTP session, the FTP client attempts to log in automatically using the current username and password stored in the system's credentials or configuration.

However, in some scenarios, you may want to suppress this auto-login feature, such as when you're connecting to multiple FTP servers with different credentials or when you want to prompt for credentials interactively.

**-d** Enables debugging, displaying all ftp commands passed between the client and server.

Debugging mode is useful for troubleshooting FTP connections and interactions between the client and server.

**-g** Disables filename globbing, which permits the use of wildcard characters in local file and path names.

Using the -g flag disables filename globbing, treating wildcard characters as literal characters rather than special characters for pattern matching.

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

### **Strengths:**

**Simplicity and Compatibility:** FTP is widely supported and easy to use, making it compatible with a wide range of systems and platforms. It's built into most operating systems and can be accessed through command-line clients or graphical interfaces.

**Efficiency:** FTP can efficiently transfer large files or directories between systems, making it suitable for bulk file transfers (GeekforGeeks, 2024).

**No Encryption Overhead:** In its basic form, FTP doesn't impose encryption overhead, which means it can be faster than encrypted protocols like SFTP (SSH File Transfer Protocol) for transferring large files (Christensson, 2015).

### **Weaknesses:**

**Lack of Encryption:** One of the significant weaknesses of FTP is its lack of encryption. FTP transmits data, including usernames, passwords, and file contents, in plaintext, making it vulnerable to eavesdropping attacks. This lack of encryption compromises the confidentiality of data transferred via FTP.

**Data Integrity Concerns:** FTP does not inherently provide mechanisms to ensure the integrity of transferred data. Without cryptographic measures like checksums or digital signatures, there's a risk that data could be tampered with during transit, compromising its integrity.

<https://www.geeksforgeeks.org/file-transfer-protocol-ftp-in-application-layer/>

**Authentication Vulnerabilities:** FTP relies on basic username/password authentication, which is susceptible to brute-force attacks, sniffing, or man-in-the-middle attacks. Weak or easily guessable passwords can lead to unauthorized access to FTP servers, impacting both confidentiality and integrity (GeekforGeeks, 2024).



## Impact on the CIA triad

According to Ham (2021), the concept of the CIA triad (Confidentiality, Integrity, and Availability) has been a core principle of information security, and later cybersecurity, since its inception. It essentially breaks down cybersecurity into three key aspects:

**Confidentiality:** Only authorised users can access sensitive information.

**Integrity:** We can be confident that information hasn't been tampered with.

**Availability:** Authorised users can access the information they need, when they need it.

**Impact on confidentiality:** FTP's lack of encryption means that data transferred via FTP is vulnerable to interception by unauthorised parties. This compromises the confidentiality of sensitive information, potentially leading to data breaches or leaks.

**Impact on integrity:** FTP lacks built-in mechanisms to verify data integrity during transfer. This means files can be tampered with in transit, leading to corruption or manipulation.

**Impact on availability:** While FTP excels in efficient file transfer operations, its availability can be impacted by denial-of-service (DoS) attacks. These attacks can disrupt or limit access to the service.

## 5) Secure network protocol

Based on the information about FTP and its security vulnerabilities, the most appropriate secure protocol to use for exchanging data would be SFTP (SSH File Transfer Protocol). SFTP provides encryption and secure authentication, addressing the weaknesses of FTP.

SFTP is a protocol that runs over SSH (Secure Shell) and provides secure file transfer capabilities. Unlike FTP, which uses separate control and data channels (and is inherently insecure), SFTP uses a single encrypted channel for both commands and data, providing strong encryption and authentication.

The images below depicts a basic client server application.

### Server side

The command **sudo service ssh start** was used to start the ssh server. Thereafter, **sudo service ssh status** was the command used to display the status of the ssh server.

```
tc@server:/$ sudo service ssh start
tc@server:/$ sudo service ssh status
• ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2024-05-23 08:35:55 UTC; 7h ago
     Docs: man:sshd(8)
           man:sshd_config(5)
   Process: 917 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 950 (sshd)
     Tasks: 1 (limit: 9347)
    Memory: 2.9M
       CPU: 335ms
   CGroup: /system.slice/ssh.service
           └─950 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

### Client side

The command **sftp user@server** was used to connect to the sftp server

Once the password was entered, the directory was changed to **/var/log** using the **cd** command.

The command **"get"** followed by the filename to be downloaded was then used to obtain the targeted file.

```
(kali@kali)-[~]
└─$ sftp tc@192.168.31.131
tc@192.168.31.131's password:
Connected to 192.168.31.131.
sftp> cd /var/log
sftp> get scanWHOIS.txt
Fetching /var/log/scanWHOIS.txt to scanWHOIS.txt
scanWHOIS.txt                                100% 2781    2.4MB/s   00:00
sftp> █
```

## SFTP addressing the CIA triad

**Confidentiality:** SFTP runs over SSH, providing encryption for all the data that is transmitted between the client and the server, ensuring that all data is protected. SFTP encrypts both commands and data which prevents external parties from accessing the information

**Integrity:** As SFTP runs over SSH, it runs integrity checks for data transmitted between the client and the server, ensuring that files have not been altered during the transit. SFTP's use of encryption helps maintain data integrity.

**Availability:** SFTP services are still vulnerable to DOS attacks but SSH's security features reduce the risks.

## 6) Conclusion

In conclusion, this cyber security project has shed light on the significant advantages of using SFTP over FTP, particularly in terms of enhancing the CIA Triad - Confidentiality, Integrity, and Availability. By leveraging encryption, integrity checks, and SSH's robust security features, SFTP provides a more secure environment for transferring files, safeguarding sensitive information from unauthorised access and tampering.

However, it's crucial to recognize that while the infrastructure provided by SFTP greatly enhances security, there are additional key elements that play a vital role in ensuring the health of the CIA Triad. Factors such as password masking and proper permission settings (chmod levels) for users are essential aspects of maintaining a secure environment and preserving the integrity and availability of data.

Moreover, beyond the cybersecurity realm, this project has also underscored the benefits of effective and efficient scripting practices. Utilising variables in naming automated files aids in differentiating output files, streamlining organisation and management. Additionally, incorporating functions in scripting facilitates code modularity and readability, simplifying the script writing process and enhancing maintainability.

In essence, this project has not only deepened understanding of the advantages of SFTP and its impact on the CIA Triad but has also highlighted the importance of holistic security practices and efficient scripting techniques in ensuring robust cybersecurity measures.

## 7) Recommendations

Following the drafting, execution and evaluation of the automation script, numerous points of recommendation could be highlighted.

- 1) After execution of the nmap command, it often took very long for the nmap command to be completed. This is likely due to the fact that nmap scans all 65,535 ports of the given ip address. An alternative, though not as exhaustive, is utilising the '-F' flag for nmap. The -F flag allows for a fast scan by only scanning the most common 100 ports rather than all 65,535 ports. This reduces the amount of time needed for the scan but unfortunately loses out on exhaustiveness.

The said command would be given as

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S nmap -F $url > /var/log/scanWHOIS.txt "
```

- 2) As previously mentioned, it was important that the automation script should have a command to revert the permissions of /var and the /var/log folder. This would be the appropriate command to be added to the back of the script in order to protect both the data availability and data integrity.

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S chmod 775 /var /var/log"
```

- 3) As more automated scans would be done in the future, the user will face difficulty organising the full detailed logs when there are numerous detailed logs of the same name "scanWHOIS.txt" and "scanNMAP.txt". Hence, amendments could be made to the following original:

```
#NMAP scanning and saving into /var/log
echo "Now checking the nmap data of the given URL/Address, saving scanned whois data to /var/log as scanNMAP.txt"
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S nmap -O $url > /var/log/scanNMAP.txt $url" #uses sshpass
nmapTIME=$(date +"%H:%M:%S") #creates Variable for time of nmap scan
nmapDATE=$(date +"%Y-%m-%d") #creates Variable for date of nmap scan
nmapDAY=$(date +"%A") #creates Variable for day of nmap scan
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "echo "An nmap scan was completed on $nmapDAY, $nmapDATE at $nmapTIME"
sleep 1
```

Into the following amended script

```
# Get current date, time, and day
nmapTIME=$(date +%H-%M-%S) # Time of nmap scan
nmapDATE=$(date +%Y-%m-%d) # Date of nmap scan
nmapDAY=$(date +%A) # Day of nmap scan

# Define output file name with date, time, day, and scan type
output_file="/var/log/scanNMAP_${nmapDATE}_${nmapTIME}_${nmapDAY}_${url}_nmap.txt"

# Information string
echo "Now checking the nmap data of the given URL/Address, saving scanned whois data to ${output_file}"

# Run nmap command remotely and save output to the defined file
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S nmap -O $url > $output_file"
```

This way, the date, time and day is taken before the nmap scan is done so that it can be included in the output file name.

- 4) After drafting the script, it is observed that there were numerous times whereby certain parts of commands were repeated while using sshpass. This could be easily avoided with a more effective and efficient use of functions in the script. In all instances of using sshpass, the section

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip"
```

Was constantly repeated numerous times.

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" echo "Connected to server!" sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S chmod 777 /var /var/log"
```

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" 'sudo -S touch /var/log/scanned.log && sudo -S chmod 777 /var/log/scanned.log'
```

```
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S whois $url > /var/log/scanWHOIS.txt "
sshpass -p "$remotepass" ssh "$remoteuser@$remoteip" "sudo -S nmap $url > /var/log/scanNMAP.txt "
```

This could have been simply replaced by the use of a function with reference below

```
1 #/bin/bash
2
3 function loginssh()
4 {
5     sshpass -p tc ssh tc@192.168.31.131 "$@"
6 }
7
8
9 loginssh "sudo -S chmod 777 /var /var/log"
10 loginssh 'sudo -S touch /var/log/scanned.log && sudo -S chmod 777 /var/log/scanned.log'
11 loginssh "sudo -S whois 192.168.31.130 > /var/log/scanWHOIS.txt "
12 loginssh "sudo -S nmap 192.168.31.130 > /var/log/scanNMAP.txt "
13
```

In this example, the variables were removed and substituted with the actual inputs (variables will still work fine on this improvement). From the above example. Using

functions easily allows us to improve our scripting tidiness as there won't be repeats of the same exhaustive and lengthy commands.

- 5) Finally, in order to prevent prying eyes from viewing the user's password, the input of the user's password could use the flag -s for the read command. This means that it does not display the characters as the user types.

# References

BasuMallick , C. (2022) *What Is FTP (File Transfer Protocol)? Definition, Uses, and Best Practices for 2022*, *Spiceworks.com*. Available at: <https://www.spiceworks.com/tech/networking/articles/what-is-ftp/> (Accessed: 24 May 2024).

Christensson, P. (2015) *FTP, Definition - What is the FTP protocol?* Available at: <https://techterms.com/definition/ftp> (Accessed: 24 May 2024).

Comer, D. E., & Stevens, D. L. (2000). *Internetworking with TCP/IP. Volume 3: Client-Server Programming and Applications (BSD Socket Version)*. Prentice Hall.

Communications, G. (2018) *Introduction to linux file permissions & attributes: Chmod, Globo.Tech*. Available at: <https://www.globo.tech/learning-center/linux-file-permissions-attributes-chmod/> (Accessed: 24 May 2024).

GeeksforGeeks (2024) *File transfer protocol (FTP) in Application Layer*, *GeeksforGeeks*. Available at: <https://www.geeksforgeeks.org/file-transfer-protocol-ftp-in-application-layer/> (Accessed: 24 May 2024).

Ham, J.V. (2021) 'Toward a better understanding of "cybersecurity"', *Digital Threats: Research and Practice*, 2(3), pp. 1–3. doi:10.1145/3442445.

IETF. (2020). *File Transfer Protocol (FTP): HOOKUP Command for TLS Negotiation*. [Online] IETF.org. <https://datatracker.ietf.org/doc/rfc959/> (Accessed on [23rd May 2024]).

Perldoc. (n.d.). Perl Documentation. [Online]. Available at: <https://perldoc.perl.org/> (Accessed: May 22, 2024).

SolarWinds (2022) *List of FTP commands for windows: SERV-U, Serv*. Available at: <https://www.serv-u.com/ftp-server-windows/commands> (Accessed: 24 May 2024).