Project:
Network Remote Control

Prepared by:
Priscilla Ang
S18

Table of content

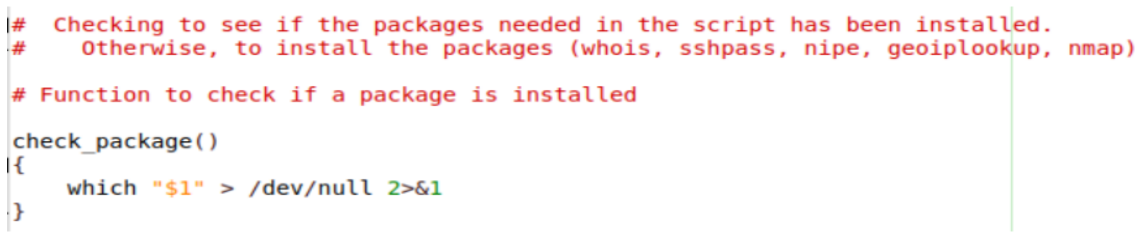
Introduction

In today's digital era, it's crucial to have efficient and secure methods for exchanging files over networks. One protocol that has been essential for this purpose is the File Transfer Protocol (FTP). FTP has been widely used for transferring files between computers, forming a fundamental part of internet communication for many years. It follows a simple client-server model and uses separate channels for controlling and transferring data. However, despite its widespread use, FTP's basic design has raised concerns about security vulnerabilities, especially in an age where protecting data privacy and integrity is crucial. This report aims to explore FTP's fundamental workings, including its mechanisms, strengths, weaknesses, and its impact on network security. Additionally, we'll discuss alternative solutions like Secure File Transfer Protocol (SFTP) to highlight advancements in secure file transfer technologies. Through this investigation, we seek to gain a deeper understanding of FTP and its role in the broader context of network protocols.

Scope 1: Script automation

1. Check and install packages

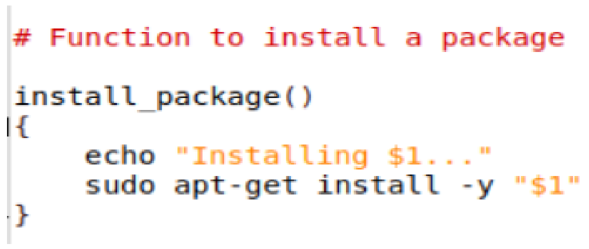
In the initial phase, we will be running a check on the machine to see if the commands used in the script have been installed in the machine. If any of these commands are found to be absent, the script will automatically initiate their installation. Given the multitude of commands to be checked, we have established both a "check package" function and an "installation package" function for streamlined validation and setup.



```
#  Checking to see if the packages needed in the script has been installed.
#    Otherwise, to install the packages (whois, sshpass, nipe, geoiplookup, nmap)

# Function to check if a package is installed

check_package()
{
    which "$1" > /dev/null 2>&1
}
```

Figure 1.1 Function for checking of packages

The 'which' command helps find where a specific command is located on the system. Here, '$1' refers to the first command provided when running the script. We're also directing any errors to '/dev/null' to keep the output clean and free from unnecessary messages.



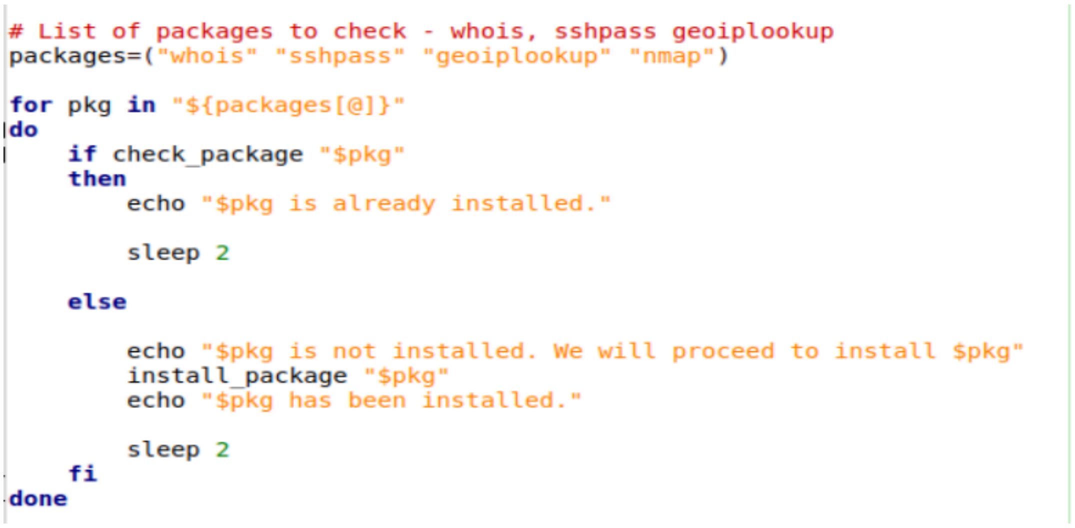
```
# Function to install a package

install_package()
{
    echo "Installing $1..."
    sudo apt-get install -y "$1"
}
```

Figure 1.2 Function for installation of packages

In this script, we utilise several command-line tools: 'geoiplookup', 'whois', 'sshpass', 'nipe', and 'nmap'.

'Geoiplookup' assists in determining the geographical location of an IP address, providing details such as the country and city, by leveraging the GeoIP database. 'Whois' serves to retrieve information about the owner of a domain name or an IP address through a query and response protocol. 'Sshpass' enables non-interactive SSH password authentication, a useful utility in scripts to automate SSH connections without requiring manual password input. 'Nipe' is a tool specifically designed for configuring and controlling IPTables to route all network traffic through the Tor network, enhancing privacy and anonymity. Finally, 'Nmap' stands as a versatile network scanning tool, allowing users to discover hosts and services on a computer network by sending packets and analysing the responses, aiding in network reconnaissance and security assessments.

In the script, a 'for-loop' iterates over each item listed in the 'packages' variable to check for their presence in the system. For instance, if the 'nipe' package is found to be installed, the script will output a message indicating that 'nipe' is already installed. However, if 'nipe' or any other package listed in the 'packages' variable is not found, the script will proceed to automatically install it. This ensures that all required packages are available on the system.

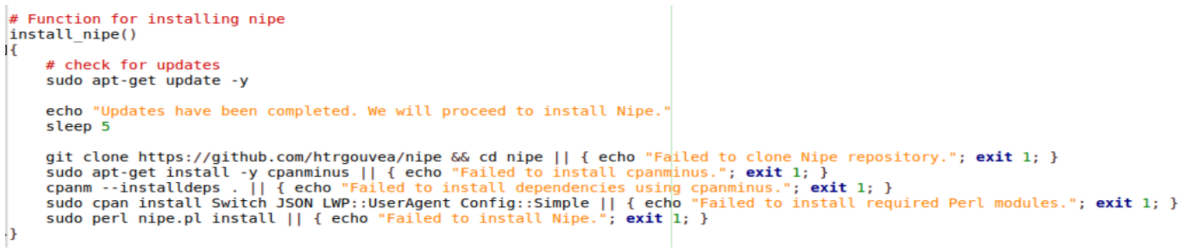


```
# List of packages to check - whois, sshpass geoiplookup
packages=("whois" "sshpass" "geoiplookup" "nmap")

for pkg in "${packages[@]}"
do
    if check_package "$pkg"
    then
        echo "$pkg is already installed."

        sleep 2

    else

        echo "$pkg is not installed. We will proceed to install $pkg"
        install_package "$pkg"
        echo "$pkg has been installed."

        sleep 2
    fi
done
```

Figure 1.3 For loop function of checking of packages.

For the installation of 'nipe', the process begins by cloning the package from its GitHub repository using the 'git clone' command. This command creates a local copy of the specified repository or branch. Following this, 'cpanminus' is installed using the 'sudo apt-get install' command. 'sudo' is used to execute commands with the security privileges of another user, allowing administrative access. The '-y' flag in the 'apt-get install' command instructs the system to automatically respond "yes" to all prompts, enabling non-interactive installation by bypassing user input. 'Cpanminus' (also known as 'cpan') is a command-line tool designed for managing Perl modules, including those required by 'nipe'. It facilitates the acquisition, extraction, building, and installation of modules from the Comprehensive Perl Archive Network (CPAN).

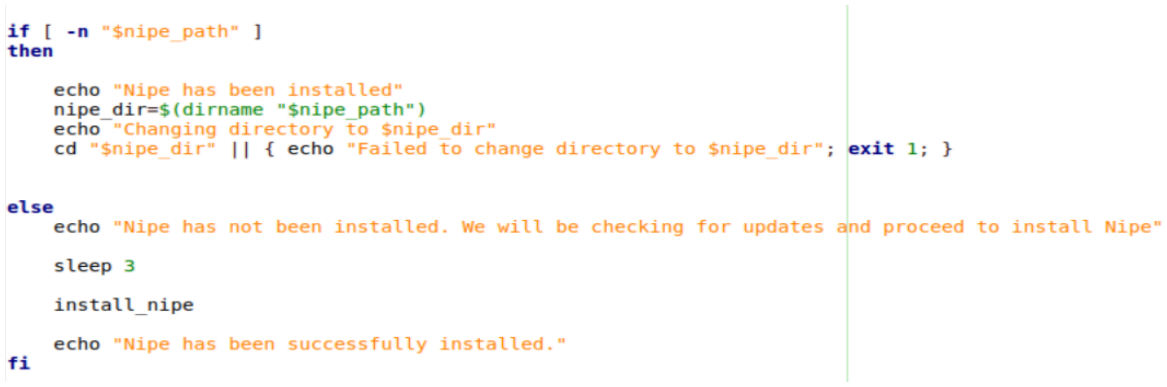
After cloning the repository and installing 'cpanminus', we proceed to install additional Perl modules required by 'nipe'. These modules include 'Switch', which furnishes a switch statement functionality absent in Perl by default. 'JSON' is employed for handling JavaScript Object Notation data, facilitating the encoding of Perl data structures into JSON and the decoding of JSON data into Perl data structures. 'LWP::UserAgent' serves as a Perl module functioning as a web client, enabling the initiation of web requests, retrieval of web pages, and interaction with web services. Lastly, 'Config::Simple' is used for reading and writing configuration files in a straightforward format.



```
# Function for installing nipe
install_nipe()
{
    # check for updates
    sudo apt-get update -y

    echo "Updates have been completed. We will proceed to install Nipe."
    sleep 5

    git clone https://github.com/htrgouvea/nipe && cd nipe || { echo "Failed to clone Nipe repository."; exit 1; }
    sudo apt-get install -y cpanminus || { echo "Failed to install cpanminus."; exit 1; }
    cpanm --installdeps . || { echo "Failed to install dependencies using cpanminus."; exit 1; }
    sudo cpan install Switch JSON LWP::UserAgent Config::Simple || { echo "Failed to install required Perl modules."; exit 1; }
    sudo perl nipe.pl install || { echo "Failed to install Nipe."; exit 1; }
}
```

Figure 1.4 Function for installing 'nipe'

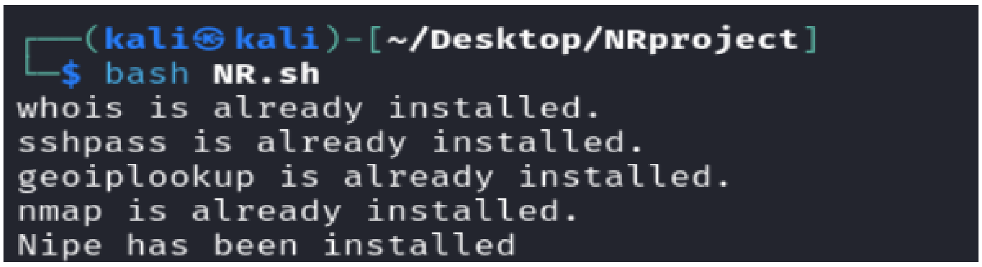
Once all prerequisites are installed, we proceed with the installation of 'nipe.pl', which necessitates root privileges. 'Nipe' is a Perl-based engine designed to transform the Tor network into the default network gateway, thereby anonymizing all internet traffic by routing it through Tor. To ensure robust error handling, each command in the installation process is followed by a check for success. If any command fails, an error message is displayed, and the script exits to prevent subsequent steps from executing without the necessary dependencies.



```
if [ -n "$nipe_path" ]
then

    echo "Nipe has been installed"
    nipe_dir=$(dirname "$nipe_path")
    echo "Changing directory to $nipe_dir"
    cd "$nipe_dir" || { echo "Failed to change directory to $nipe_dir"; exit 1; }

else
    echo "Nipe has not been installed. We will be checking for updates and proceed to install Nipe"

    sleep 3

    install_nipe

    echo "Nipe has been successfully installed."
fi
```

Figure 1.5 if-else statement to check if 'nipe' has been installed.

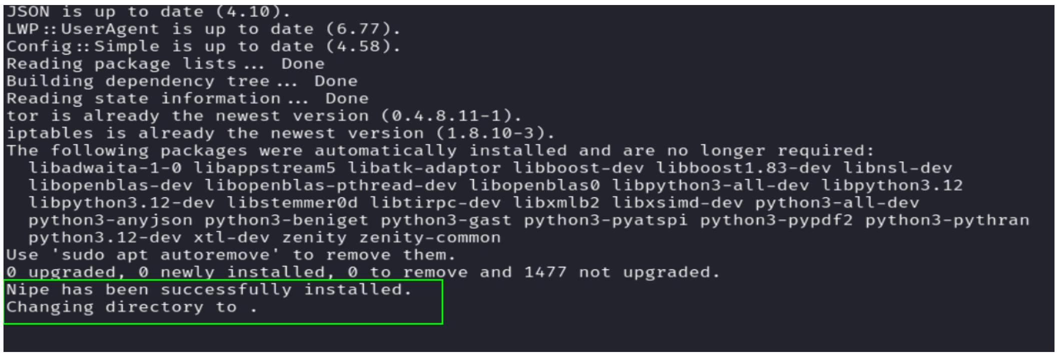
This if-else statement evaluates whether the variable 'nipe_path' is empty. The '-n' flag checks if the length of the string stored in '$nipe_path' is greater than zero. If '$nipe_path' is not empty, indicating that 'nipe' has been installed on the machine, the condition returns true. In this case, the script extracts the directory path from 'nipe_path' and changes to that directory. However, if 'nipe_path' is empty, implying that 'nipe' is not installed, the script proceeds to call the 'install_nipe' function (figure 1.4) to initiate the installation process.



```
┌──(kali㉿kali)-[~/Desktop/NRproject]
└─$ bash NR.sh
whois is already installed.
sshpass is already installed.
geoiplookup is already installed.
nmap is already installed.
Nipe has been installed
```

Figure 1.5 Machine echoing the packages that have been installed.

Once 'nipe' is successfully installed, the script changes the directory to the location where 'nipe' has been installed. This step is essential because 'nipe' requires the user to be in the same directory where it is installed in order to execute properly. By changing to the installation directory, the script ensures that subsequent commands related to 'nipe' will be executed from the correct location.

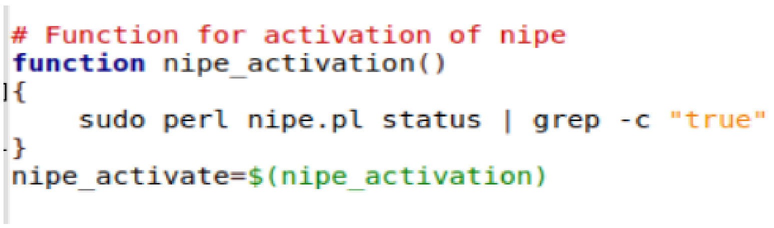


```
JSON is up to date (4.10).
LWP::UserAgent is up to date (6.77).
Config::Simple is up to date (4.58).
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
tor is already the newest version (0.4.8.11-1).
iptables is already the newest version (1.8.10-3).
The following packages were automatically installed and are no longer required:
  libadwaita-1-0 libappstream5 libatk-adaptor libboost-dev libboost1.83-dev libnsl-dev
  libopenblas-dev libopenblas-pthread-dev libopenblas0 libpython3-all-dev libpython3.12
  libpython3.12-dev libstemmer0d libtirpc-dev libxmlb2 libxsimd-dev python3-all-dev
  python3-anyjson python3-beniget python3-gast python3-pyatspi python3-pypdf2 python3-pythran
  python3.12-dev xtl-dev zenity zenity-common
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 1477 not upgraded.
Nipe has been successfully installed.
Changing directory to .
```

Figure 1.6 Machine echo that ‘nipe’ has been installed and changing directory to where ‘nipe’ is.

2. Connecting to ‘nipe’

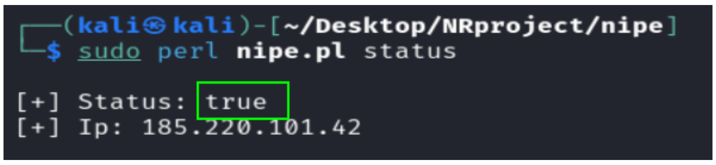
To establish a connection with 'nipe', the script first verifies whether 'nipe' is already activated. If it's not activated, the script proceeds to activate it. This ensures that 'nipe' is properly configured and ready for use before attempting to establish any connections.



```
# Function for activation of nipe
function nipe_activation()
{
    sudo perl nipe.pl status | grep -c "true"
}
nipe_activate=$(nipe_activation)
```

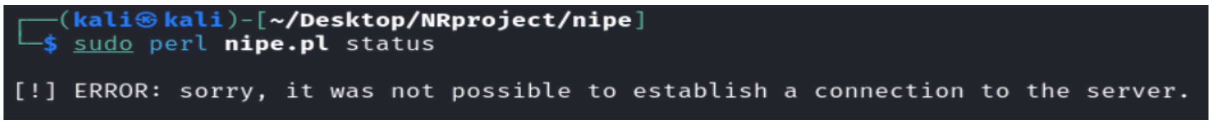
Figure 2.1 Function for activation of ‘nipe’

The script employs the command ‘sudo perl nipe.pl status’ to retrieve the status of 'nipe'. Subsequently, it utilises ‘grep’ to search for the string "true" within the obtained output (figure 2.1). By specifying the ‘-c’ flag, ‘grep’ counts the occurrences of lines containing "true" and provides this count as output. If 'nipe' is active, the count returned by grep -c "true" will be greater than 0 (figure 2.2). Conversely, if 'nipe' is inactive, the count will be 0 since "true" will not be found in the output (figure 2.3). The script captures this output and assigns it to the variable 'nipe_activate' for further use.



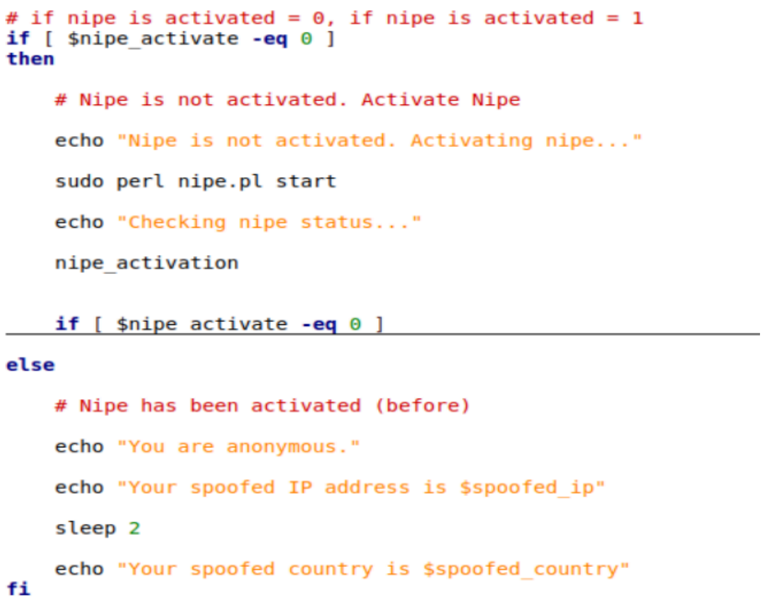
```
┌──(kali㉿kali)-[~/Desktop/NRproject/nipe]
└─$ sudo perl nipe.pl status

[+] Status: true
[+] Ip: 185.220.101.42
```

Figure 2.2 ‘nipe’ is connected, returning with a “true” status and the spoofed IP address it is connected to is shown



```
┌──(kali㉿kali)-[~/Desktop/NRproject/nipe]
└─$ sudo perl nipe.pl status

[!] ERROR: sorry, it was not possible to establish a connection to the server.
```

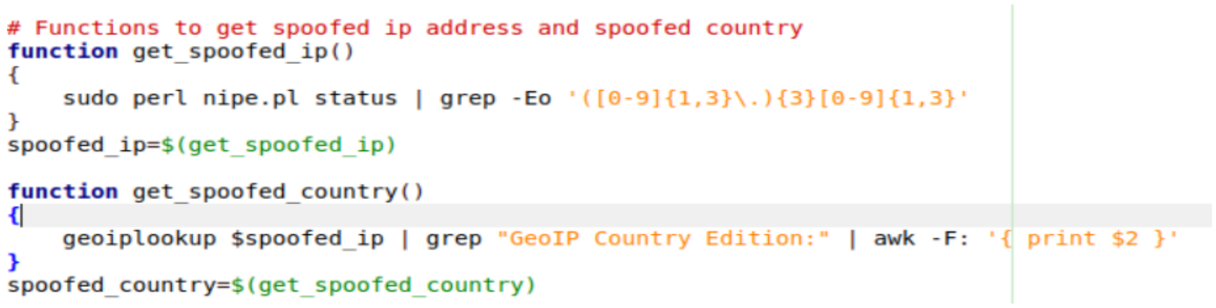
Figure 2.3 ‘nipe’ is not connected.



```
# if nipe is activated = 0, if nipe is activated = 1
if [ $nipe_activate -eq 0 ]
then

    # Nipe is not activated. Activate Nipe

    echo "Nipe is not activated. Activating nipe..."

    sudo perl nipe.pl start

    echo "Checking nipe status..."

    nipe_activation


    if [ $nipe activate -eq 0 ]

else

    # Nipe has been activated (before)

    echo "You are anonymous."

    echo "Your spoofed IP address is $spoofed_ip"

    sleep 2

    echo "Your spoofed country is $spoofed_country"
fi
```

Figure 2.4 Statement to check if ‘nipe’ is activated

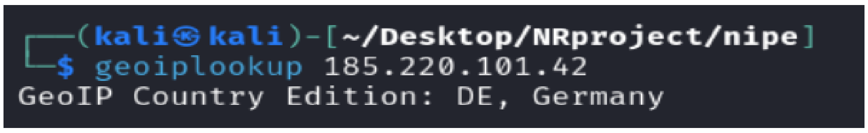
The if-else statement in the script (figure 2.4) is designed to verify the activation status of 'nipe'. If the variable '$nipe_activate' is equal to 0, indicating that 'nipe' is not activated, the script proceeds to activate it. It then checks its status again to ensure successful activation. On the other hand, if 'nipe' is already activated, the script informs the user that they are anonymous, displaying the spoofed IP and spoofed country.



```
# Functions to get spoofed ip address and spoofed country
function get_spoofed_ip()
{
    sudo perl nipe.pl status | grep -Eo '([0-9]{1,3}\.){3}[0-9]{1,3}'
}
spoofed_ip=$(get_spoofed_ip)

function get_spoofed_country()
{
    geoiplookup $spoofed_ip | grep "GeoIP Country Edition:" | awk -F: '{ print $2 }'
}
spoofed_country=$(get_spoofed_country)
```

Figure 2.5 Functions to retrieve the spoofed ip address and country

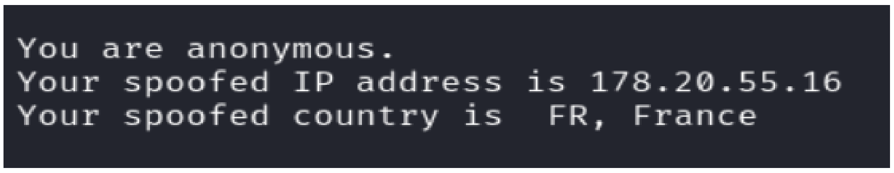
In Figure 2.2, when 'nipe' is connected, the spoofed IP address is displayed below the status. To extract the spoofed IP address from the status output, 'grep' is used with the '-E' flag to enable extended regular expressions and the '-o' flag to extract only the matching part of the line. The regular expression '([0-9]{1,3}.){3}[0-9]{1,3}' matches an IP address pattern, allowing us to filter out and extract the IP address from the status output. This extracted IP address is stored in a variable named 'spoofed_ip'. Once the spoofed IP address is obtained, we can proceed to look up the corresponding country of the spoofed IP address (as shown in Figure 2.6).



```
┌──(kali㉿kali)-[~/Desktop/NRproject/nipe]
└─$ geoiplookup 185.220.101.42
GeoIP Country Edition: DE, Germany
```

Figure 2.6 Command to get spoofed country from IP address

In the process described in Figure 2.6, the 'grep' command is initially used to filter out the line containing the country edition, ensuring that only the correct line is selected. Then, the 'awk' command is employed with the '-F:' option to specify the colon (:) as the field separator. By doing so, the line is split into two columns based on the colon separator, and the 'awk' command prints out the second column, which corresponds to the country code (e.g., DE). This effectively extracts the country information from the output, providing the desired country name (e.g., Germany).

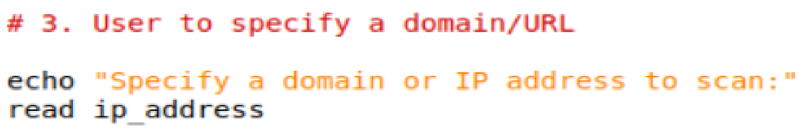


```
You are anonymous.
Your spoofed IP address is 178.20.55.16
Your spoofed country is  FR, France
```

Figure 2.7 Final output of the spoofed IP address and country.

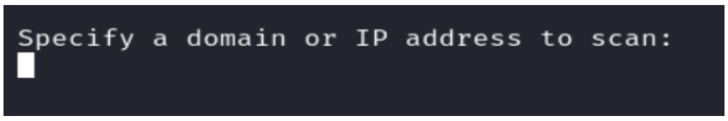
3. Specify domain/URL

The script initiates a user prompt to input either a domain name or an IP address. The information provided by the user is then stored in a variable named 'ip_address'. This variable serves as a reference to the domain or IP address throughout the script's execution.



```
# 3. User to specify a domain/URL

echo "Specify a domain or IP address to scan:"
read ip_address
```

Figure 3.1 Command for user to specify an IP address



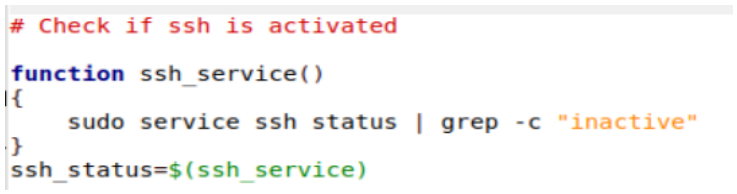
```
Specify a domain or IP address to scan:
```

Figure 3.2 Prompt for user to key in a domain or IP address

4. Connect to remote server via SSH

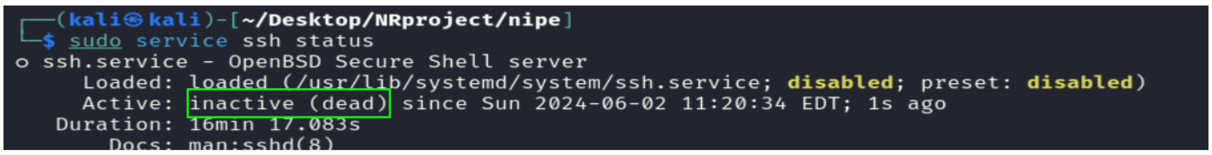
4.1 Checking of SSH status

The script employs a comparable approach to verify and establish the connection for 'ssh'. Utilising the command 'sudo service ssh status', it examines whether the SSH service is active or inactive. The 'grep' command, coupled with the '-c' flag, quantifies the occurrences of the term "inactive" within the output.



```
# Check if ssh is activated

function ssh_service()
{
    sudo service ssh status | grep -c "inactive"
}
ssh_status=$(ssh_service)
```

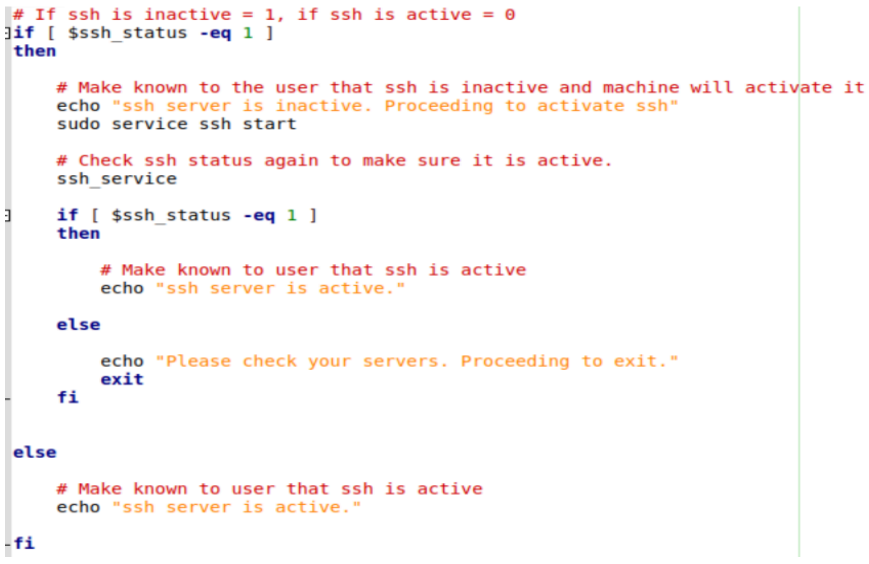
Figure 4.1 Function to check for ssh service



```
┌──(kali㉿kali)-[~/Desktop/NRproject/nipe]
└─$ sudo service ssh status
○ ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: disabled)
     Active: inactive (dead) since Sun 2024-06-02 11:20:34 EDT; 1s ago
   Duration: 16min 17.083s
       Docs: man:sshd(8)
```

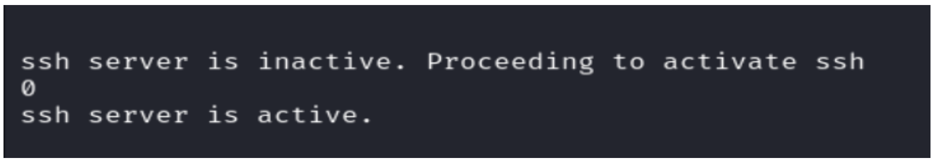
Figure 4.2 Ssh status inactive (dead)

If the status is inactive, denoted by a count of 1, the script proceeds to activate SSH using the command 'sudo service ssh start'. Conversely, if the count is different from 1, indicating that SSH is already running, the script notifies the user accordingly.



```
# If ssh is inactive = 1, if ssh is active = 0
if [ $ssh_status -eq 1 ]
then

    # Make known to the user that ssh is inactive and machine will activate it
    echo "ssh server is inactive. Proceeding to activate ssh"
    sudo service ssh start

    # Check ssh status again to make sure it is active.
    ssh_service

    if [ $ssh_status -eq 1 ]
    then

        # Make known to user that ssh is active
        echo "ssh server is active."

    else

        echo "Please check your servers. Proceeding to exit."
        exit
    fi


else

    # Make known to user that ssh is active
    echo "ssh server is active."

fi
```

Figure 4.2 If-else statement for the condition that ssh is not switched on.

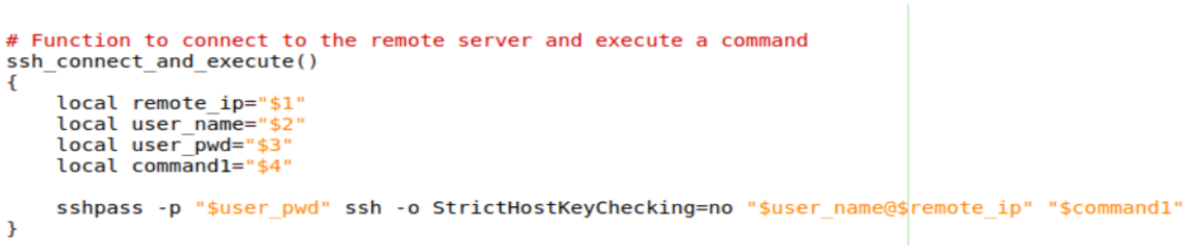


```
ssh server is inactive. Proceeding to activate ssh
0
ssh server is active.
```

Figure 4.3 Terminal message of activating ssh.

4.2 Connection through sshpass

The 'ssh_connect_and_execute' function facilitates connection to a remote server and execution of a specified command without initiating an interactive shell session. This is achieved using the 'sshpass' utility.



```
# Function to connect to the remote server and execute a command
ssh_connect_and_execute()
{
    local remote_ip="$1"
    local user_name="$2"
    local user_pwd="$3"
    local command1="$4"

    sshpass -p "$user_pwd" ssh -o StrictHostKeyChecking=no "$user_name@$remote_ip" "$command1"
}
```

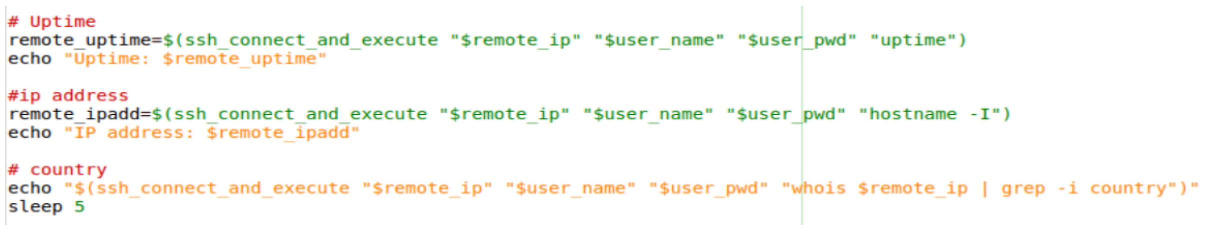
Figure 4.4 Function to connect to remote server and executing a command

The 'sshpass' utility facilitates non-interactive password authentication for SSH connections, commonly employed in automated scripts. Its '-p' flag enables the direct specification of the SSH password within the command line, allowing 'sshpass' to transmit the password to the 'ssh' command automatically. This automation streamlines the login process, eliminating the need for manual input. However, it's important to note that this approach may present security vulnerabilities, as the password becomes visible in the command line and accessible to other users on the system.

Additionally, the '-o' flag in the 'ssh' command specifies options, such as 'StrictHostKeyChecking'. When set to 'no', as in '-o StrictHostKeyChecking=no', the script configures SSH to bypass the usual host key verification process. Consequently, SSH will automatically accept any host keys provided by the server, adding them to the user's known_hosts file and allowing connections without further confirmation.

4.3 Collecting data from remote server

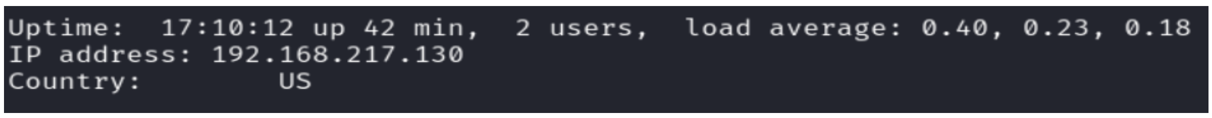
Once connected, the script will collect data from the remote server, specifically its IP address, country, and open ports.



```
# Uptime
remote_uptime=$(ssh_connect_and_execute "$remote_ip" "$user_name" "$user_pwd" "uptime")
echo "Uptime: $remote_uptime"

#ip address
remote_ipadd=$(ssh_connect_and_execute "$remote_ip" "$user_name" "$user_pwd" "hostname -I")
echo "IP address: $remote_ipadd"

# country
echo "$(ssh_connect_and_execute "$remote_ip" "$user_name" "$user_pwd" "whois $remote_ip | grep -i country")"
sleep 5
```

Figure 4.5 Commands for data collection of the remote server

'Uptime' is a metric that measures system reliability and stability. It represents the percentage of time that a system remains continuously operational and available for use. Additionally, it helps monitor how long the system has been running since the last reboot or shutdown.



```
Uptime:  17:10:12 up 42 min,  2 users,  load average: 0.40, 0.23, 0.18
IP address: 192.168.217.130
Country:        US
```

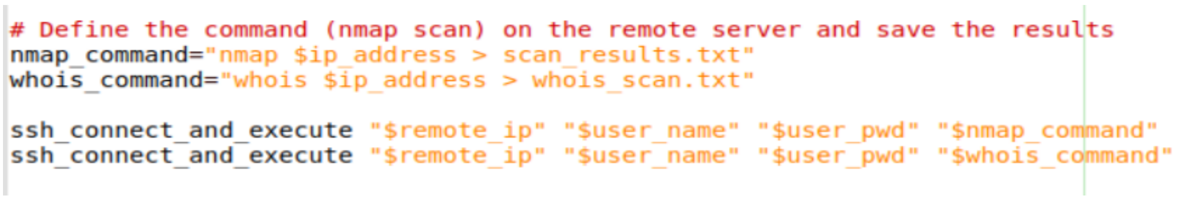
Figure 4.6 Output of uptime, IP address and country of the IP address.

The output of the 'uptime' command (figure 4.6) shows several key pieces of information. First, it displays the current time of the computer in 'hh:mm:ss' format. Next, it shows the total time the system has been running, represented in days, hours, and minutes. Additionally, it indicates the number of active users currently logged into the system. Finally, it provides the average system load over the last one, five, and fifteen minutes.

4.4 Scanning of ports and retrieving data from remote server

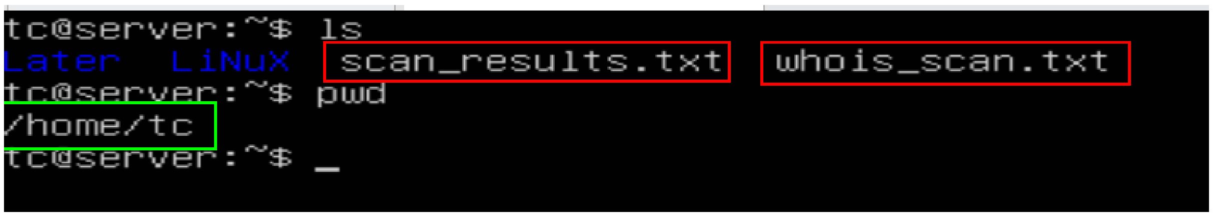
To scan the ports and retrieve data for the user-specified domain, the script uses the 'nmap' command, which scans the ports of the specified domain stored in the 'ip_address' variable (as illustrated in figure 3.2). The scan results are saved in the scan_results.txt file on the remote server (figure 4.8). Similarly, the 'whois' command retrieves domain information, and its results are saved in the whois_scan.txt file.

These commands are executed on the remote server using the 'ssh_connect_and_execute' function.



```
# Define the command (nmap scan) on the remote server and save the results
nmap_command="nmap $ip_address > scan_results.txt"
whois_command="whois $ip_address > whois_scan.txt"

ssh_connect_and_execute "$remote_ip" "$user_name" "$user_pwd" "$nmap_command"
ssh_connect_and_execute "$remote_ip" "$user_name" "$user_pwd" "$whois_command"
```

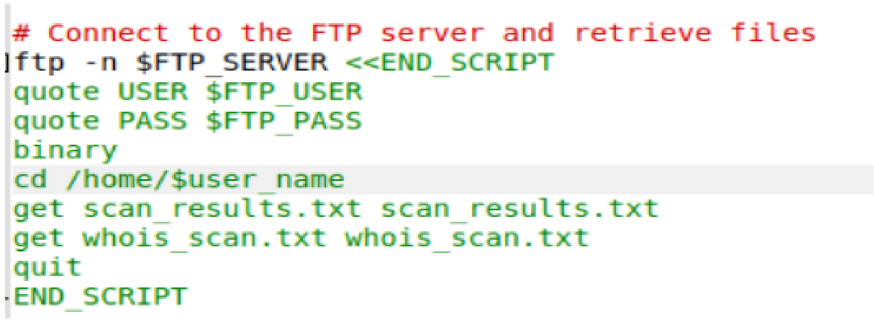
Figure 4.7 Command for 'nmap' scan and 'whois' on the remote server



```
tc@server:~$ ls
Later   LiNuX   scan_results.txt   whois_scan.txt
tc@server:~$ pwd
/home/tc
tc@server:~$ _
```

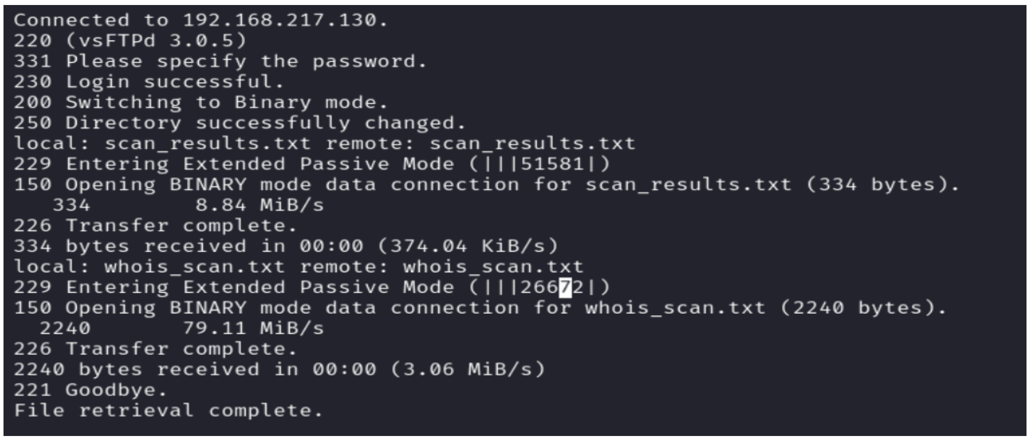
Figure 4.8 Scan results (Red rectangle) and its file path (green rectangle) saved on the remote server

The script utilises FTP connectivity to fetch scan results stored on a remote server. The '-n' flag employed with FTP suppresses automatic login, necessitating manual authentication. Utilising the quote command, specific directives or arguments are conveyed to the FTP server. In this instance, 'quote USER' and 'quote PASS' are employed to furnish the server with the username and password for authentication purposes. The 'binary' command designates binary mode for file transfers, ensuring that files are transmitted without alteration, preserving their integrity. Subsequently, the 'get' command facilitates the retrieval of a file from the remote server, enabling its transfer to the user's local machine. The command channel remains open until the 'QUIT' command is enforced to disconnect the server.



```
# Connect to the FTP server and retrieve files
ftp -n $FTP_SERVER <<END_SCRIPT
quote USER $FTP_USER
quote PASS $FTP_PASS
binary
cd /home/$user_name
get scan_results.txt scan_results.txt
get whois_scan.txt whois_scan.txt
quit
END_SCRIPT
```

Figure 4.9 Script to connect and retrieve files from FTP server

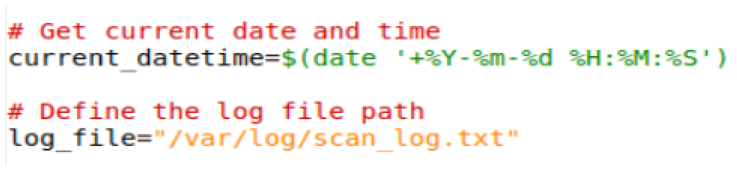


```
Connected to 192.168.217.130.
220 (vsFTPd 3.0.5)
331 Please specify the password.
230 Login successful.
200 Switching to Binary mode.
250 Directory successfully changed.
local: scan_results.txt remote: scan_results.txt
229 Entering Extended Passive Mode (|||51581|)
150 Opening BINARY mode data connection for scan_results.txt (334 bytes).
   334        8.84 MiB/s
226 Transfer complete.
334 bytes received in 00:00 (374.04 KiB/s)
local: whois_scan.txt remote: whois_scan.txt
229 Entering Extended Passive Mode (|||26672|)
150 Opening BINARY mode data connection for whois_scan.txt (2240 bytes).
  2240       79.11 MiB/s
226 Transfer complete.
2240 bytes received in 00:00 (3.06 MiB/s)
221 Goodbye.
File retrieval complete.
```

Figure 4.10 Terminal display of successful connection and retrieval of files from the remote server through FTP connection

4.5 Logging scan details into /var/log

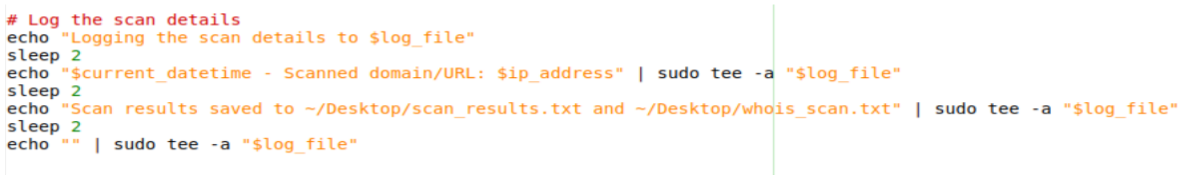
The scan results are logged into a file located in the /var/log directory.



```
# Get current date and time
current_datetime=$(date '+%Y-%m-%d %H:%M:%S')

# Define the log file path
log_file="/var/log/scan_log.txt"
```

Figure 4.11 Command for date, time and defining the log file path

The ‘date’ command is used to display the current date and time on the system. The format '+%Y-%m-%d %H:%M:%S' specifies how the date and time should be displayed. The /var/log directory contains log files from the operating system, services, and various applications running on the system.

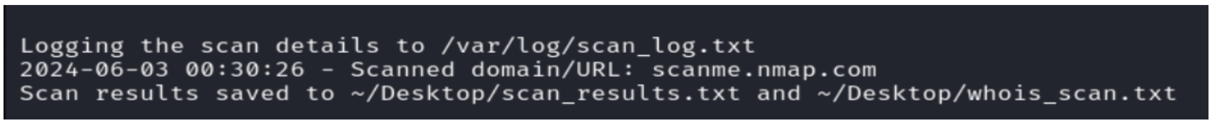


```
# Log the scan details
echo "Logging the scan details to $log_file"
sleep 2
echo "$current_datetime - Scanned domain/URL: $ip_address" | sudo tee -a "$log_file"
sleep 2
echo "Scan results saved to ~/Desktop/scan_results.txt and ~/Desktop/whois_scan.txt" | sudo tee -a "$log_file"
sleep 2
echo "" | sudo tee -a "$log_file"
```

Figure 4.12 Logging of the scan details

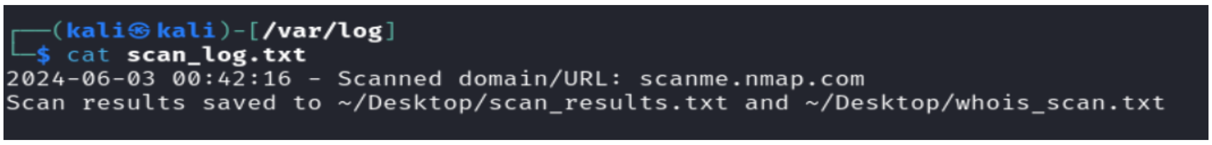
The ‘tee’ command reads from the standard input and writes to both the standard output and one or more files. This allows the input from a command to be written to a file while still being displayed on the terminal. The ‘-a’ option tells ‘tee’ to append the output to the end of the specified file, rather than overwriting it.

These results, including the current date and time as well as the scan outcomes, are saved into a file labelled ‘scan_log.txt’ (figure 4.14). Simultaneously, the same message is displayed on the terminal (figure 4.13) for user reference.



```
Logging the scan details to /var/log/scan_log.txt
2024-06-03 00:30:26 - Scanned domain/URL: scanme.nmap.com
Scan results saved to ~/Desktop/scan_results.txt and ~/Desktop/whois_scan.txt
```

Figure 4.13 Terminal message on the scan logged



```
┌──(kali㉿kali)-[/var/log]
└─$ cat scan_log.txt
2024-06-03 00:42:16 - Scanned domain/URL: scanme.nmap.com
Scan results saved to ~/Desktop/scan_results.txt and ~/Desktop/whois_scan.txt
```

Figure 4.14 Same output saved in the Scanned log in the /var/log folder.

Scope 2: Network research and monitoring

The script employs FTP (File Transfer Protocol) to establish a connection with the remote server and download files. FTP is a client-server protocol used for transferring files between computers over TCP/IP (Transmission Control Protocol/Internet Protocol) networks. Its main function is to enable the management and transfer of large files on remote servers, providing users with the capability to access and share files from remote systems. This functionality is crucial for tasks such as website management, data exchange, and backup solutions.

Request for Comments (RFC)

The FTP protocol is outlined in several RFCs, with RFC 959 being the most significant. Published by the Internet Engineering Task Force (IETF) in October 1985, RFC 959 defines the FTP standard, detailing the command structure, data transmission modes, authentication mechanisms, error handling procedures, and other key aspects of the protocol. It provides a thorough explanation of FTP operations, including the establishment of control and data connections, the format of commands and responses, and the functional behaviour of FTP clients and servers. RFC 959 served as the main reference for implementing FTP in software and networking systems. Despite being updated by newer standards, it remains a foundational document that laid the groundwork for file transfer mechanisms widely used in computer networks.

Additional updates to FTP standards include RFC 2640, which addresses the internationalisation of FTP, and RFC 2228, which introduces security enhancements such as encrypted passwords, authentication tokens, and session encryption using SSL/TLS protocols. RFC 2773 describes further encryption of file data and the FTP command channel using KEA and SKIPJACK. RFC 7151 adds a new FTP command to enable FTP clients and servers to identify individual virtual hosts on an FTP server.

Key features of FTP

As the primary purpose of FTP is to facilitate file transfer between a client and a server, it supports both uploading (sending files from the client to the server) and downloading (retrieving files from the server to the client). Backup services and individual users often use FTP to backup or replicate data from one location to a secured backup server running FTP services. It is also commonly used for accessing shared web hosting and cloud services, providing a mechanism to load data onto a remote system. This ensures a reliable method for sharing and distributing files across different systems and networks, simplifying the process of managing files on web servers for developers and administrators.

A significant advantage of using FTP is its ability to transfer a large number or large size of files efficiently. If a user needs to send gigabytes of data in one session, FTP can handle this process quickly, minimising inefficiencies and saving time. Additionally, FTP can generate directory listings, allowing users to view files within a directory easily. This feature makes it straightforward for users to navigate the server. Furthermore, FTP supports the resume functionality, enabling interrupted transfers to be resumed from the point of failure, which is crucial for transferring large files over unstable network connections.

Regarding security, FTP traditionally uses only a username and password to prevent unauthorised access, as it was originally designed for non-confidential purposes, such as exchanging non-sensitive data between remote employees. Nowadays, FTP can be configured to allow anonymous access, permitting users to download files without requiring a username and password. This is commonly used for public file repositories.

FTP supports both binary and ASCII (American Standard Code for Information Interchange) transfer modes. Binary mode is primarily used for transferring data types such as images or other graphic files that do not contain line feeds, preserving the integrity of binary files. ASCII mode is used for text-based files such as HTML, text documents, and other similar files, reducing the risk of file corruption and allowing users to transfer a variety of file types freely.

FTP sessions operate in either active or passive mode. In active mode, after a client initiates a session via a command channel request, the server creates a data connection back to the client and begins transferring data. In passive mode, the server uses the command channel to send the client the information needed to open a data channel. Since passive mode has the client initiating all connections, it is more effective across firewalls and network address translation gateways.

How FTP works

FTP operates on a client-server model, where the client initiates a connection to an FTP server. In this interaction model, a program (the client) sends a request to another program (the server) and waits for a response. The client is the program users utilise to communicate and interact with the FTP server, while the server stores all the files.

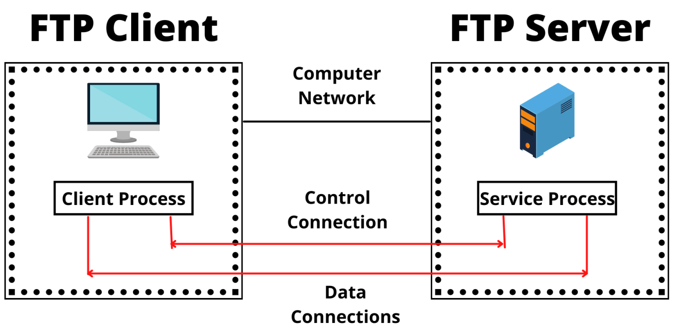


Figure 5.1 Basic FTP client-server model (image: developingdaily)

FTP communication channels

FTP employs two distinct connections to manage file transfers: the control connection and the data connection. The control channel is used for transmitting control information such as user identification, password, and commands to change the remote directory. This connection is initiated on port 21. The data connection, which is used for transfer of files, is initiated on port 20. The server port used for this channel depends on the communication mode being employed, either active or passive.

In active mode, the FTP client establishes a control connection and sends the server a command through this channel, specifying the port the server should connect to. The server then responds by connecting to that port. Active mode is effective in environments without firewalls or with firewalls that understand the FTP protocol and can automatically open the necessary ports between clients and servers.

In passive mode, both data and control connections are initiated by the FTP client to the server. The server provides a port number that the client should connect to, and the client then connects to that port on the server. This mode is considered "firewall-friendly" as it functions well in environments with firewalls, enhancing security and reliability. In passive mode, the server listens passively on a port and no inbound connections from the internet to the clients are initiated, making it more secure.

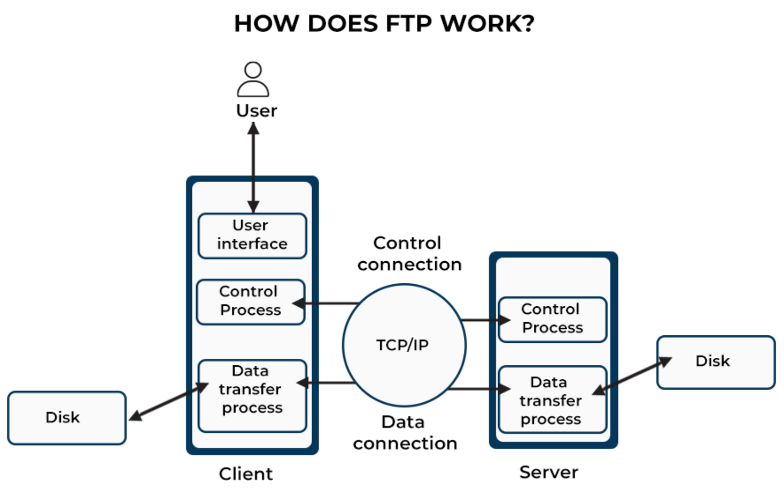


Figure 5.2 Image showing how FTP works and its communication channels (image: spiceworks)

Data representation and storage

Transmission modes

FTP transfers files through three transmission modes: stream mode, block mode, and compressed mode. Stream mode is the default mode, where data is transmitted from FTP to TCP as a continuous stream of bytes. TCP then segments this data into smaller packets. In block mode, data is transmitted from FTP to TCP in distinct blocks, each prefixed by a 3-byte header. The first byte of this header contains information about the block, referred to as the description block, while the remaining two bytes specify the block size. Compressed mode is employed for transferring large files, which can be challenging due to size constraints over the internet. This mode compresses the files into smaller sizes, facilitating more efficient transmission.

File structures

FTP supports three types of file structures: file structure, record structure, and page structure. The file structure is the simplest, consisting of a sequence of data bytes joined together without any internal organisation. It serves as the default structure for files if not otherwise specified. The record structure organises files into a collection of data records arranged in sequence. Both file and record structures are suitable for handling "text" files. The page structure is designed for transmitting files with interruptions or gaps, where the file comprises multiple independently indexed pages. Each page is sent with a page header containing fields such as header length, page index, data length, and page type, facilitating error-free storage and retrieval of files with varying page sizes and associated information.

Data types

FTP allows users to specify the representation type for data handling during transfer, offering options such as ASCII, EBCDIC, image, and local types. The default representation, ASCII, is universally accepted by all FTP implementations and is ideal for transferring text files. The EBCDIC type is optimised for efficient transfer between hosts that use EBCDIC for their internal character representation. On the other hand, the image type is tailored for the efficient storage, retrieval, and transfer of binary data, making it suitable for files like images or executables. Finally, the local type enables data transfer in bytes of a specified size, providing flexibility in data transfer configurations.

Sequence of events

FTP initiates its operation by establishing a control connection, employing TCP as the transport protocol on port 21. Prior to transmitting data segments between devices, TCP employs a 3-way handshake mechanism to establish a connection and ensure synchronisation. This process, known as the TCP 3-way handshake, involves a series of communications between two devices aimed at establishing a dependable network connection and guaranteeing error-free data transmission.

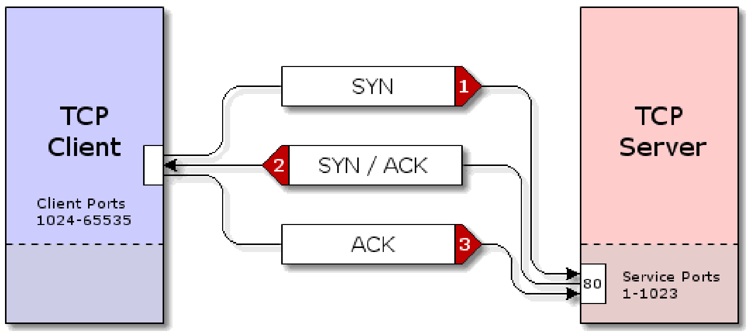


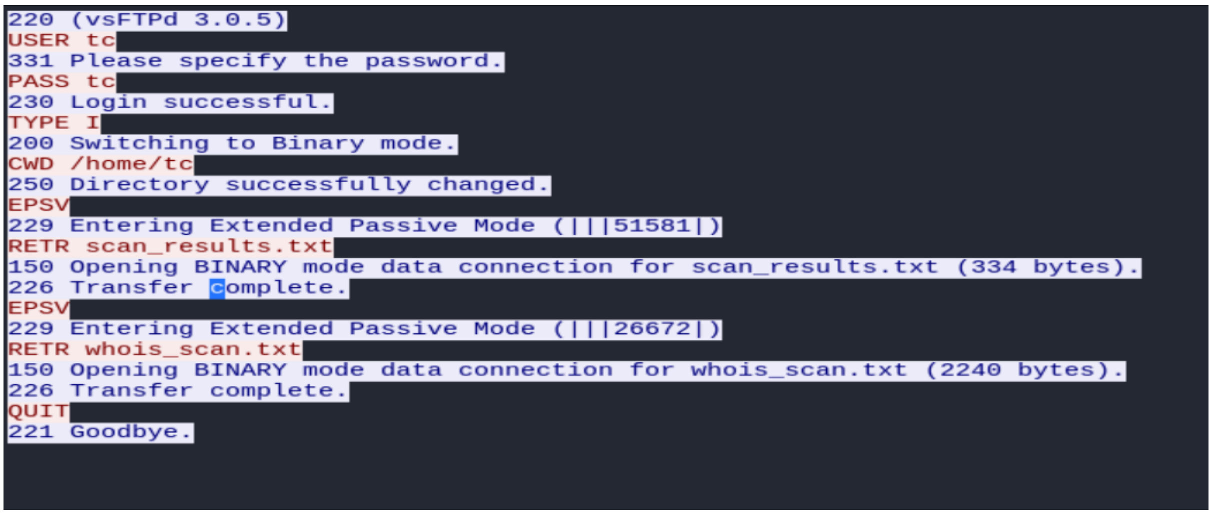
Figure 5.3 TCP 3-way handshake (image: medium)

The TCP 3-way handshake, a fundamental process in establishing network connections, comprises three sequential steps. Initially, the client dispatches a SYN (synchronise) packet to the server, followed by the server's response with a SYN-ACK (synchronise-acknowledge) packet, acknowledging the client. Subsequently, the client confirms the server's acknowledgment by transmitting an ACK (acknowledge) packet, thereby completing the handshake and establishing a reliable connection conducive to data transfer. This method ensures synchronisation, reliability, and flow control in TCP/IP communications, facilitating ordered and error-checked data transmission.

Following connection establishment, the client proceeds with user authentication, sending the username to the server, which subsequently requests the password. Upon successful authentication, the server returns a '230' response signifying successful login, or '530' in the case of authentication failure.

Client commands for directory navigation, file listing, upload, and download operations are executed subsequently, with the server providing corresponding responses for each command. File retrieval is initiated with the 'RETR' command, while file upload is initiated with the 'STOR' command, prompting the server to establish a data connection for file transfer. Upon completion, the server closes the data connection and issues a '226' completion response. For each file transfer operation, the server orchestrates the opening, transfer, and subsequent closure of the data connection.

To conclude the session, the client transmits the 'QUIT' command, prompting the server to terminate the control connection and issue a '221' message signifying session termination.



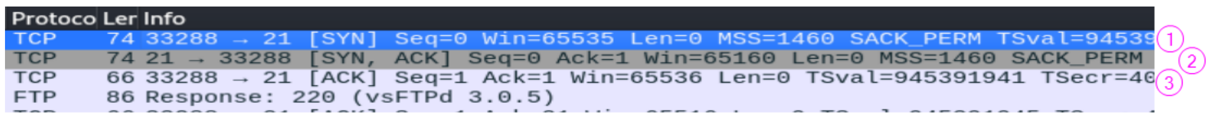
```
220 (vsFTPd 3.0.5)
USER tc
331 Please specify the password.
PASS tc
230 Login successful.
TYPE I
200 Switching to Binary mode.
CWD /home/tc
250 Directory successfully changed.
EPSV
229 Entering Extended Passive Mode (|||51581|)
RETR scan_results.txt
150 Opening BINARY mode data connection for scan_results.txt (334 bytes).
226 Transfer complete.
EPSV
229 Entering Extended Passive Mode (|||26672|)
RETR whois_scan.txt
150 Opening BINARY mode data connection for whois_scan.txt (2240 bytes).
226 Transfer complete.
QUIT
221 Goodbye.
```

Figure 5.4 Sequence of events following the TCP stream through the FTP protocol.

Message Exchange process

FTP entails a sequence of commands and responses shared between the client and server, facilitating the transfer of files and data via distinct control and data connections.

Initiating the process, the FTP client initiates contact with the FTP server on port 21 to establish a TCP connection. A successful TCP 3-way handshake is then established to facilitate communication between the client and server.

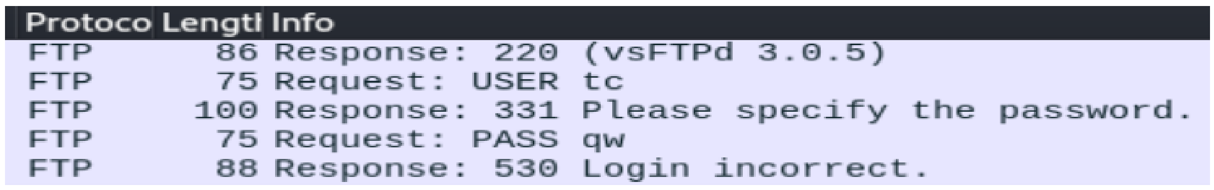


```
Protoco Ler Info
TCP     74 33288 → 21 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 SACK_PERM TSval=94539  ①
TCP     74 21 → 33288 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM    ②
TCP     66 33288 → 21 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=945391941 TSecr=40  ③
FTP     86 Response: 220 (vsFTPd 3.0.5)
```

Figure 5.5 TCP 3-way handshake for connection to FTP server.

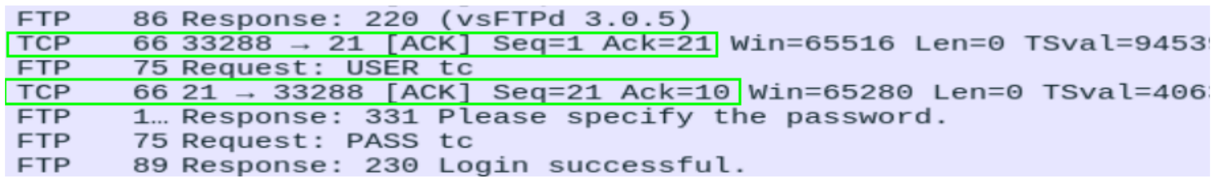
1 - The client computer initiates a connection to the server via a packet with only the SYN flag.
2 - The server replies to this request with both the ACK and SYN flag.
3 - The client responds back to the server with a single ACK packet, thus establishing a connection between the client and the server.

Following this initial connection establishment, the client proceeds to authenticate itself by submitting the user's username, eliciting a server response indicating a request for the password with the message '331 please specify password'. Upon the client's submission of the password via the 'PASS' command, the server acknowledges the successful login with a '230 login successful' response. Conversely, an unsuccessful login attempt prompts the server to convey a '530 login unsuccessful' message.



```
Protoco Lengtl Info
FTP         86 Response: 220 (vsFTPd 3.0.5)
FTP         75 Request: USER tc
FTP        100 Response: 331 Please specify the password.
FTP         75 Request: PASS qw
FTP         88 Response: 530 Login incorrect.
```

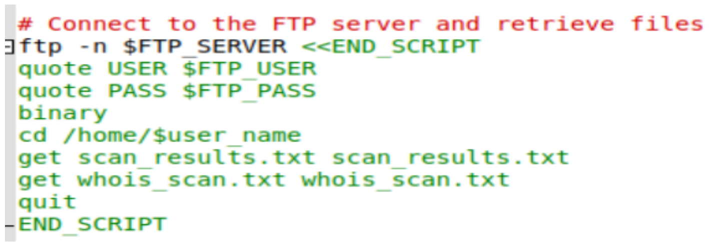
Figure 5.6 Unsuccessful login.



```
FTP    86 Response: 220 (vsFTPd 3.0.5)
TCP    66 33288 → 21 [ACK] Seq=1 Ack=21 Win=65516 Len=0 TSval=9453
FTP    75 Request: USER tc
TCP    66 21 → 33288 [ACK] Seq=21 Ack=10 Win=65280 Len=0 TSval=406
FTP    1… Response: 331 Please specify the password.
FTP    75 Request: PASS tc
FTP    89 Response: 230 Login successful.
```

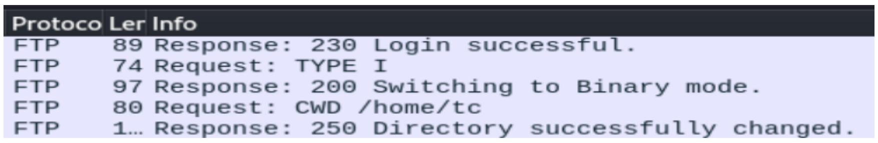
Figure 5.7 FTP protocol: user authentication.
TCP protocol: ACK packet to confirm receipt from the server (green rectangle)

Given that FTP operates over the TCP protocol, each server response triggers an acknowledgment (ACK) packet from the client, serving as a fundamental aspect of TCP's reliability mechanism. This acknowledgment mechanism ensures the orderly and accurate reception of each data segment, upholding data integrity throughout the communication process.



```
# Connect to the FTP server and retrieve files
ftp -n $FTP_SERVER <<END_SCRIPT
quote USER $FTP_USER
quote PASS $FTP_PASS
binary
cd /home/$user_name
get scan_results.txt scan_results.txt
get whois_scan.txt whois_scan.txt
quit
END_SCRIPT
```

Figure 5.8 Commands input by the user.

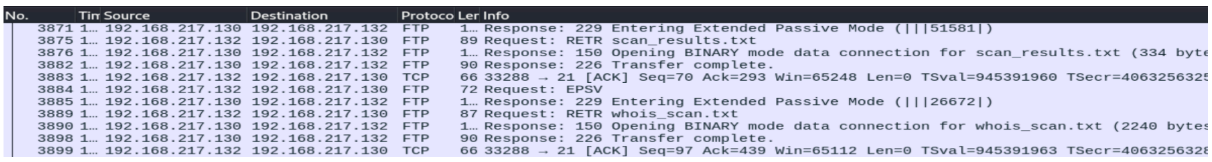


```
Protoco Ler Info
FTP     89 Response: 230 Login successful.
FTP     74 Request: TYPE I
FTP     97 Response: 200 Switching to Binary mode.
FTP     80 Request: CWD /home/tc
FTP     1… Response: 250 Directory successfully changed.
```

Figure 5.9 Commands executed on FTP by the user.

As per the user's instructions depicted in figure 5.8, the FTP protocol initially switches to TYPE I, commonly referred to as binary mode. Subsequently, a command is issued to navigate to the user's home directory, denoted in Wireshark as CWD /home/tc, with 'tc' representing the username. Upon execution, the server confirms the successful directory change.

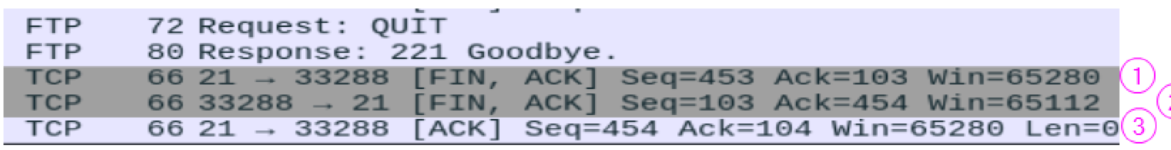
When tasked with commands necessitating data transfer, the server proceeds to establish a data connection. It begins by entering an extended passive mode (EPSV), transmitting solely the port number, with the client inferring that it connects to the original IP address. Subsequently, the server initiates a TCP data connection to the client (packet 3876 in figure 5.9) to facilitate data transfer. Following the successful completion of the data transfer, the server confirms the successful transfer and terminates the data connection.



```
No.   Tir Source          Destination     Protoco Ler Info
 3871 1… 192.168.217.130 192.168.217.132 FTP     1… Response: 229 Entering Extended Passive Mode (|||51581|)
 3875 1… 192.168.217.132 192.168.217.130 FTP     89 Request: RETR scan_results.txt
 3876 1… 192.168.217.130 192.168.217.132 FTP     1… Response: 150 Opening BINARY mode data connection for scan_results.txt (334 byte
 3882 1… 192.168.217.130 192.168.217.132 FTP     90 Response: 226 Transfer complete.
 3883 1… 192.168.217.132 192.168.217.130 TCP     66 33288 → 21 [ACK] Seq=70 Ack=293 Win=65248 Len=0 TSval=945391960 TSecr=406325632
 3884 1… 192.168.217.132 192.168.217.130 FTP     72 Request: EPSV
 3885 1… 192.168.217.130 192.168.217.132 FTP     1… Response: 229 Entering Extended Passive Mode (|||26672|)
 3889 1… 192.168.217.132 192.168.217.130 FTP     87 Request: RETR whois_scan.txt
 3890 1… 192.168.217.130 192.168.217.132 FTP     1… Response: 150 Opening BINARY mode data connection for whois_scan.txt (2240 bytes
 3898 1… 192.168.217.130 192.168.217.132 FTP     90 Response: 226 Transfer complete.
 3899 1… 192.168.217.132 192.168.217.130 TCP     66 33288 → 21 [ACK] Seq=97 Ack=439 Win=65112 Len=0 TSval=945391963 TSecr=406325632
```

Figure 5.10 Wireshark reflects the connections made over the transfer of two files.

For subsequent file transfers, the server initiates another TCP data connection (as observed in packet 3890 in figure 5.9). Upon successful completion of the transfer, the server issues a confirmation response and proceeds to close the connection.



```
FTP     72 Request: QUIT
FTP     80 Response: 221 Goodbye.
TCP     66 21 → 33288 [FIN, ACK] Seq=453 Ack=103 Win=65280
TCP     66 33288 → 21 [FIN, ACK] Seq=103 Ack=454 Win=65112
TCP     66 21 → 33288 [ACK] Seq=454 Ack=104 Win=65280 Len=0
```

Figure 5.11 Terminating of session through 3-way handshake

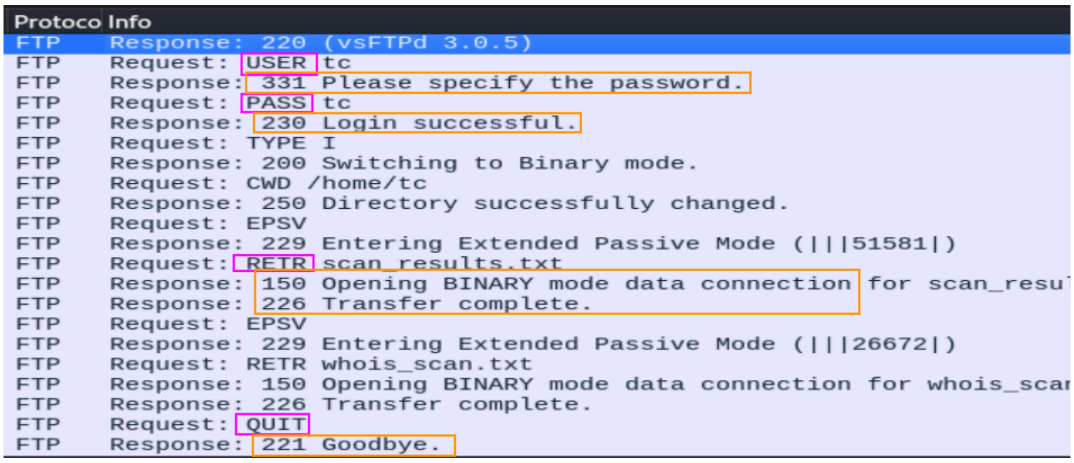
1 - The server sends a FIN packet, indicating it has finished sending data.
2 - The client replies with FIN, ACK flag.
3 - The server responds back to the client with a single ACK packet, thus terminating the connection between the client and the server.

To conclude the session, the client initiates termination by sending a 'QUIT' command, prompting the server to respond with a '221 Goodbye' acknowledgement. Subsequently, the server terminates the connection using a 4-way handshake: it sends a TCP segment with the FIN (finish) flag to signal the completion of data transmission. The client acknowledges receipt by sending a FIN, ACK flag, and the server responds with an ACK packet, acknowledging the termination of the session.

Mechanisms of FTP

Message formats

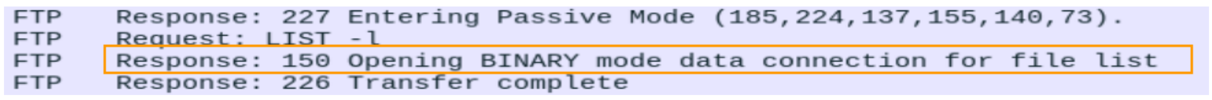
FTP employs diverse message formats to facilitate communication between the client and server throughout file transfer operations, encompassing both commands and responses. Commands, initiated by the FTP client, direct the server to execute specific tasks such as file uploads, downloads, and directory navigation. Typically conveyed as straightforward text strings, each command begins with a keyword and may be accompanied by relevant parameters. For instance, common FTP commands include "RETR" for file retrieval, "LIST" for directory listing, and "QUIT" for session termination. In contrast, responses are issued by the FTP server to acknowledge client commands and furnish feedback regarding the outcome of requested actions. Comprising a numeric code followed by a human-readable message, these responses communicate the status of the operation at hand, a topic that will be elaborated upon in subsequent sections.



```
Protoco Info
FTP     Response: 220 (vsFTPd 3.0.5)
FTP     Request: USER tc
FTP     Response: 331 Please specify the password.
FTP     Request: PASS tc
FTP     Response: 230 Login successful.
FTP     Request: TYPE I
FTP     Response: 200 Switching to Binary mode.
FTP     Request: CWD /home/tc
FTP     Response: 250 Directory successfully changed.
FTP     Request: EPSV
FTP     Response: 229 Entering Extended Passive Mode (|||51581|)
FTP     Request: RETR scan_results.txt
FTP     Response: 150 Opening BINARY mode data connection for scan_resu
FTP     Response: 226 Transfer complete.
FTP     Request: EPSV
FTP     Response: 229 Entering Extended Passive Mode (|||26672|)
FTP     Request: RETR whois_scan.txt
FTP     Response: 150 Opening BINARY mode data connection for whois_sca
FTP     Response: 226 Transfer complete.
FTP     Request: QUIT
FTP     Response: 221 Goodbye.
```

Figure 5.11 Conversations between the client and server. Commands (purple rectangle) and responses (orange rectangle)

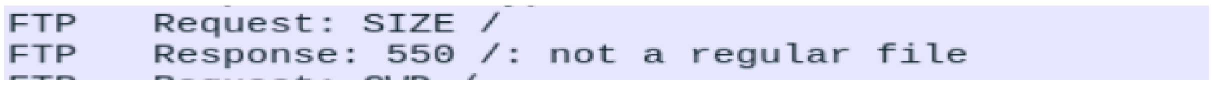
FTP uses different channels for sending control messages and transferring data. Data transfer messages specify how the data is formatted and organised when sent between the client and server during file transfers. They indicate whether the data is in ASCII or binary format and what type of data it represents (like text or images).



```
FTP    Response: 227 Entering Passive Mode (185,224,137,155,140,73).
FTP    Request: LIST -l
FTP    Response: 150 Opening BINARY mode data connection for file list
FTP    Response: 226 Transfer complete
```

Figure 5.12 Conversation between the client and FTP server on data transfer (orange rectangle)

In addition to regular responses, FTP also sends error messages when problems occur during file transfers. These error messages explain what went wrong, such as not having permission to access a file or the server rejecting the connection. They help users understand and fix problems that arise.



```
FTP    Request: SIZE /
FTP    Response: 550 /: not a regular file
```

Figure 5.13 Conversation between the client and FTP server showing an error message.

Header structure of FTP

The FTP server sends responses, known as FTP replies, to acknowledge or complete commands issued by the user. These replies serve to inform the user about the status of their commands and any errors encountered during the process. They are structured to be easily understood by both humans and programs. Each FTP reply consists of a three-digit numeric code followed by a space and a text explanation.

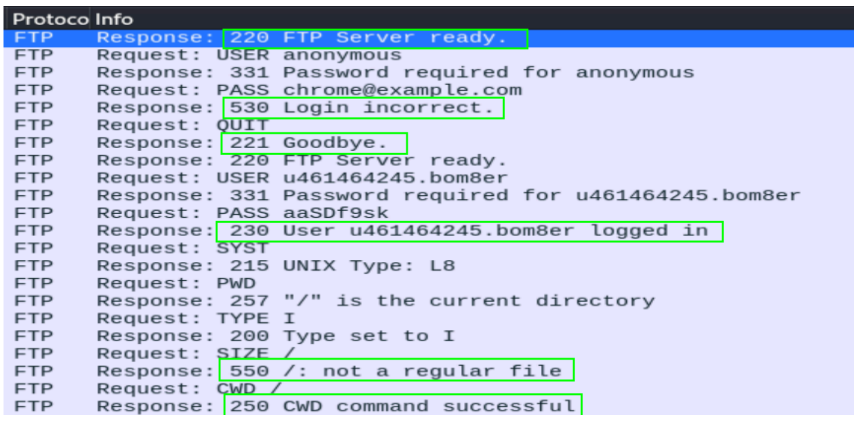
These numeric codes serve as standardised indicators, providing specific information about the outcome of the command issued by the user. Codes beginning with "1" signify responses to status inquiries, indicating a positive acknowledgment of the status command. Codes beginning with "2" denote a positive acknowledgment of the previous command or another successful action. Codes beginning with "3" indicate incomplete information, requiring further specification and input for the activity to proceed. Codes beginning with "4" signify an unsuccessful reply from the server, typically due to a temporary failure caused by a client error. Codes beginning with "5" indicate an incorrect or illegal command, signifying that the command or its parameters are invalid or incomplete, resulting in failure.

The second digit in these numeric codes categorises the response into different general categories. Codes ranging from x00 to x29 represent general-purpose replies that are not specifically assigned to other categories. Codes ending with "30" indicate primary access, typically responses to "log-in" attempts. Codes ending with "40" signify secondary access, where the server is commenting on its ability to access a secondary service. Codes ending with "5X" represent FTP results.

The final digit in these numeric codes specifies a particular message type. However, since the code is primarily designed for automated processing, it is unnecessary for every variation of a reply to have a unique number. Therefore, only the basic meanings of replies are assigned unique numbers, ensuring efficient interpretation by automated systems.

After the numeric code, there is a human-readable message that offers additional context or information regarding the status or purpose of the command. This message aims to be easily comprehensible by users or administrators, typically providing details about the action taken, any encountered errors, or instructions for further steps.

Some common FTP replies include “220 FTP Server ready”, “230 user logged in”, “226 Transfer complete” “550 Permission denied”, “221 Goodbye”.



```
Protoco Info
FTP     Response: 220 FTP Server ready.
FTP     Request: USER anonymous
FTP     Response: 331 Password required for anonymous
FTP     Request: PASS chrome@example.com
FTP     Response: 530 Login incorrect.
FTP     Request: QUIT
FTP     Response: 221 Goodbye.
FTP     Response: 220 FTP Server ready.
FTP     Request: USER u461464245.bom8er
FTP     Response: 331 Password required for u461464245.bom8er
FTP     Request: PASS aaSDf9sk
FTP     Response: 230 User u461464245.bom8er logged in
FTP     Request: SYST
FTP     Response: 215 UNIX Type: L8
FTP     Request: PWD
FTP     Response: 257 "/" is the current directory
FTP     Request: TYPE I
FTP     Response: 200 Type set to I
FTP     Request: SIZE /
FTP     Response: 550 /: not a regular file
FTP     Request: CWD /
FTP     Response: 250 CWD command successful
```

Figure 5.14 FTP structure consisting of three-code numeric digits and text messages

Flags or options that affect FTP behaviour

These general options are used together with the FTP commands that includes but not limited to:

- 4 forces ftp to only use IPv4 addresses
- 6 Forces ftp to only use IPv6 addresses
- g Disables globbing (the use of wildcard characters such as asterisks and question marks)
- i Turns off interactive prompting during multiple file transfers
- n Restrains ftp from attempting auto-login upon initial connection

- P	Sets the port number
- p	Allows the use of ftp in environments where a firewall prevents connections back to the client machine
- v	Displays all the responses from the remote server and provides data transfer statistics

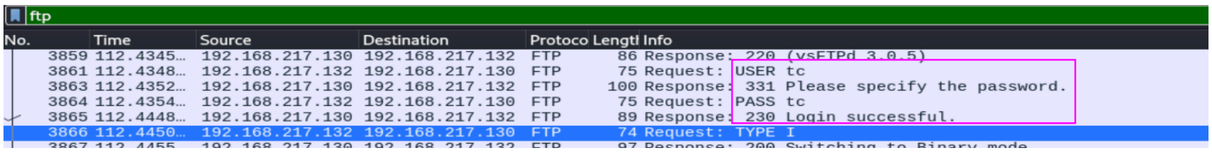
Strength and weakness of FTP

Confidentiality

FTP offers control over file access and permissions through authentication and authorization mechanisms, allowing users to authenticate themselves using credentials and granting specific rights based on configured roles or permissions. This enables organisations to enforce security policies and restrict access to sensitive files, thereby enhancing data protection and confidentiality.

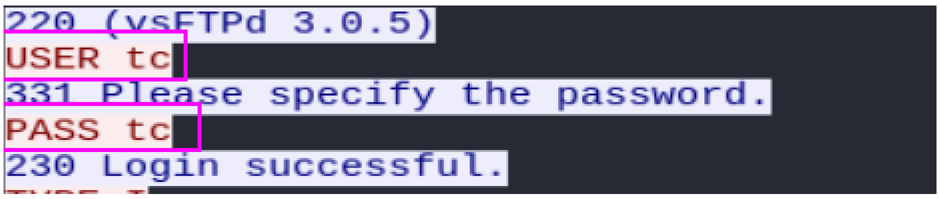
However, FTP lacks encryption, making data sent over its channels susceptible to interception and unauthorised access. As FTP uses separate connections for command and data channels, it requires multiple ports to be open on firewalls, potentially leading to compatibility issues and security vulnerabilities. In the event of a system compromise, intercepted data could be exploited, with common exploits including man-in-the-middle attacks and packet sniffing.

One of the main vulnerabilities associated with FTP is its reliance on clear-text passwords, which are passwords transmitted without encryption. Unlike more secure protocols that utilise algorithms to mask passwords, FTP does not provide protection for passwords, leaving them vulnerable to interception by malicious actors. This represents a substantial risk to sensitive information, rendering FTP unsuitable for transferring confidential data without additional encryption mechanisms such as FTPS or SFTP.



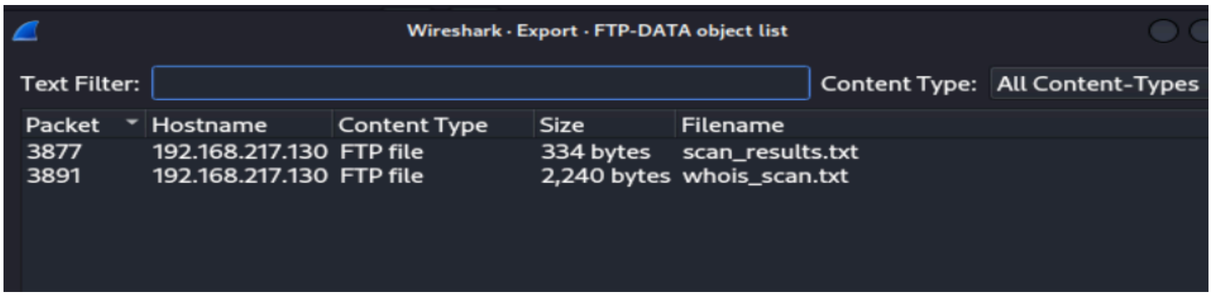
```
ftp
No.   Time        Source          Destination     Protoco Lengtl Info
 3859 112.4345…  192.168.217.130 192.168.217.132 FTP      86 Response: 220 (vsFTPd 3.0.5)
 3861 112.4348…  192.168.217.132 192.168.217.130 FTP      75 Request: USER tc
 3863 112.4352…  192.168.217.130 192.168.217.132 FTP     100 Response: 331 Please specify the password.
 3864 112.4354…  192.168.217.132 192.168.217.130 FTP      75 Request: PASS tc
 3865 112.4448…  192.168.217.130 192.168.217.132 FTP      89 Response: 230 Login successful.
 3866 112.4450…  192.168.217.132 192.168.217.130 FTP      74 Request: TYPE I
 3867 112.4455   192.168.217.130 192.168.217.132 FTP      97 Response: 200 Switching to Binary mode
```

Figure 5.15 Username and password can be seen without encryption through wireshark.



```
220 (vsFTPd 3.0.5)
USER tc
331 Please specify the password.
PASS tc
230 Login successful.
```

Figure 5.16 Username and password can be seen without encryption through the ftp stream.



Wireshark · Export · FTP-DATA object list

Text Filter: Content Type: All Content-Types

Packet	Hostname	Content Type	Size	Filename
3877	192.168.217.130	FTP file	334 bytes	scan_results.txt
3891	192.168.217.130	FTP file	2,240 bytes	whois_scan.txt

Figure 5.17 Files can be easily seen and downloaded by malicious actors.

Integrity

FTP can enhance file integrity by employing mechanisms such as Cyclic Redundancy Check (CRC) or checksums. CRC, an algorithm used to identify errors in transmitted data, generates a fixed-size value called a checksum from the data being sent. Upon reception, this checksum is recalculated to validate data integrity, minimising the risk of corruption during FTP file transfers.

Nevertheless, FTP lacks inherent measures such as digital signatures or cryptographic hashes to prevent file tampering during transmission. While CRC and checksums can be implemented for this purpose, they are not standard features of FTP, leaving transferred files potentially vulnerable to unauthorised modifications.

Availability

FTP boasts widespread support across diverse platforms, making it exceptionally accessible for file transfer requirements. Its compatibility with various systems ensures seamless file transfers regardless of the platforms involved. Moreover, FTP is renowned for its ease of implementation and use, necessitating minimal configuration and setup. Its intuitive command-line interface and standardised protocol specifications contribute to user-friendly interactions. This simplicity extends to FTP server deployment, empowering organisations to swiftly establish FTP servers for facilitating internal file sharing. Another notable advantage of FTP is its efficient handling of large file transfers.

However, FTP lacks an inherent mechanism for restricting access to and management of files on the server. Consequently, individuals armed with the server address and credentials can access and manipulate files without leaving a trace, potentially leading to data loss, corruption, or leakage. Furthermore, as FTP relies on separate connections for command and data channels, it mandates the opening of multiple ports on firewalls. This complexity in firewall configurations can render the protocol vulnerable to being blocked or interrupted by network security measures. Moreover, the reliance on multiple ports exposes the service to potential denial-of-service (DoS) attacks, disrupting file transfer operations and jeopardising service availability.

Recommendations

The utilisation of SFTP (Secure Shell (SSH) File Transfer Protocol) ensures secure file transfers via SSH, granting full access to shell accounts situated on a remote server. In contrast to FTP's use of two channels for file transfer, which poses potential security risks, SFTP operates solely through one channel, streamlining operations while upholding secure, encrypted transfers facilitated by SSH. SFTP facilitates inbound communication on port 22, which is associated with the SSH protocol, known for securely connecting to remote devices. Additionally, SFTP employs a tunnelling method for data transfer, diverging from FTP's direct transfer approach. SFTP ensures the confidentiality of transferred data by employing encryption mechanisms facilitated by SSH. Through data-in-motion encryption, sensitive information remains secure from unauthorised access or interception during transmission, preserving confidentiality.

Furthermore, SFTP incorporates public key cryptography, a cryptographic system leveraging pairs of private and public keys. This system enables data-in-motion encryption, authentication, digital signing, and data integrity mechanisms. Users can enhance security by augmenting passwords with SFTP keys, facilitating two-factor authentication (2FA). By verifying the integrity of transferred files through mechanisms like CRC (Cyclic Redundancy Check) or checksums, SFTP ensures that files remain unchanged and unaltered during transit, safeguarding against unauthorised modifications.

SFTP enhances availability by providing a reliable and stable platform for file transfers. Unlike FTP, which may encounter disruptions due to security vulnerabilities or network issues, SFTP offers robust connectivity and data transfer capabilities. Additionally, SFTP's use of a single channel for data transfer minimises potential points of failure, ensuring uninterrupted access to files and resources.

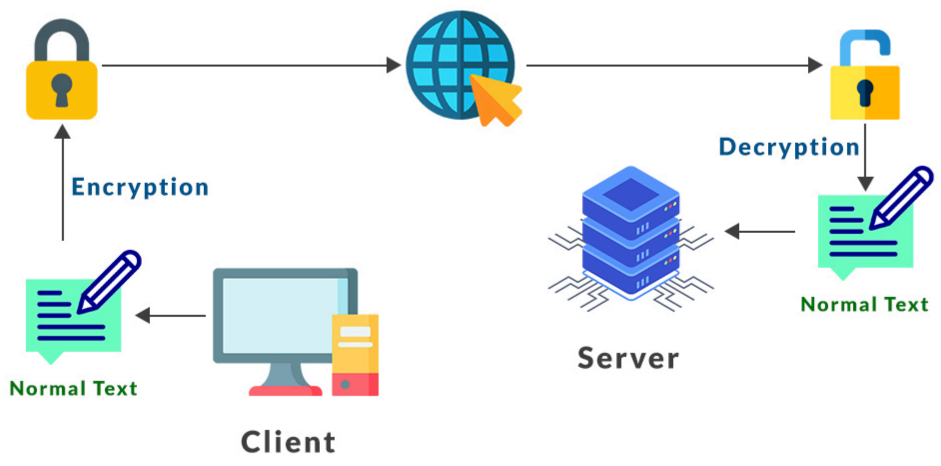


Figure 5.18 SFTP data transfer

Conclusion

Through analysing the architecture, mechanisms, and security considerations, a richer comprehension of FTP's role in facilitating file transfers and its impact on network operations has been achieved. Furthermore, delving into alternative solutions like SFTP has broadened perspectives on security-enhanced protocols, underscoring the critical importance of implementing robust measures to safeguard data integrity and confidentiality in today's interconnected digital landscape. Overall, the process of researching and documenting FTP has underscored the vital role of network protocols in shaping communication frameworks and underscored the necessity of informed decision-making in selecting appropriate protocols to meet specific organisational needs.

References:

Amoany E. (2020). SSH password automation in Linux with sshpass. [online]. RedHat. Available at: https://www.redhat.com/sysadmin/ssh-automation-sshpass (Accessed: 27 May 2024)

BasuMallick C. (2022). What is FTP (File Transfer Protocol)? Definition, uses, and best practices for 2022. [online]. Spiceworks. Available at: https://www.spiceworks.com/tech/networking/articles/what-is-ftp/ (Accessed: 31 May 2024)

Bhushan A.(1972). The file transfer protocol. [doc].ietf.Available at: https://www.ietf.org/rfc/rfc354.html (Accessed: 31 May 2024)

Cao D. (2024). Understanding SSH Stricthostkeychecking option. [online]. Howtouselinux. Available at: https://www.howtouselinux.com/post/ssh-stricthostkeychecking-option (Accessed: 27 May 2024)

GeeksforGeeks (2024).File Transfer Protocol (FTP) in Application Layer. [online]. Available at: https://www.geeksforgeeks.org/file-transfer-protocol-ftp-in-application-layer/. (Accessed: 31 May 2024)

Gite V. (2024). Sshpass - login to ssh server with a password using a shell script. [online]. Cyberciti. Available at: https://www.cyberciti.biz/faq/noninteractive-shell-script-ssh-password-provider/#:~:text=You%20can%20use%20the%20sshpass,but%20in%20non%2Dinteractive%20mode. (Accessed: 27 May 2024)

Glass V. (2024). What port does SFTP use? [online]. Jscape. Available at: https://www.jscape.com/blog/what-port-does-sftp-use#:~:text=SFTP%20uses%20port%20number%2022,S%2C%20which%20require%20multiple%20ports.(Accessed: 30 May 2024)

Goss M. (2023). 12 common network protocols and their functions explained [online]. TechTarget. Available at: https://www.techtarget.com/searchnetworking/feature/12-common-network-protocols-and-their-functions-explained (Accessed: 29 May 2024)

Htrgouvea (n.d). Nipe. [online]. Github. Available at: https://github.com/htrgouvea/nipe (Accessed: 27 May 2024)

John (2022). What is FTP? How to use it, where to use it, and the key advantages [online]. Nimbus. Available at: https://nimbushosting.co.uk/blog/what-is-ftp-how-to-use-it-where-to-use-it-and-the-key-advantages#:~:text=Probably%20the%20biggest%20advantage%20of,reducing%20inefficiencies%20and%20wasted%20time (Accessed: 29 May 2024)

Kerner S,M, Blake J. (2021) FTP (File Transfer Protocol) [online]. TechTarget. Available at: https://www.techtarget.com/searchnetworking/definition/File-Transfer-Protocol-FTP (Accessed: 29 May 2024)

Mofijul H. Md (2023). Understanding the TCP 3-way handshake in computer networking. [online]. Available at: https://www.linkedin.com/pulse/understanding-tcp-3-way-handshake-computer-networking-haque/ (Accessed: 31 May 2024)

N.A (n.d.). Git Guides - Git Clone. [online]. Github. Available at: https://github.com/git-guides/git-clone (Accessed: 27 May 2024)

N.A (2023). Host-key authentication [online]. Trellix. Available at: https://docs.trellix.com/bundle/ax_sag/page/UUID-0b24b229-e981-0acc-5afb-f1e94620c042.html#:~:text=With%20strict%20host%2Dkey%20checking,Central%20Management%20System%20appliance%20sends. (Accessed: 27 May 2024)

N.A (n.d). How to use binary or ascii mode [online]. Enterprisedt. Available at: https://enterprisedt.com/products/edtftpjssl/doc/manual/html/howtotransfermodes.html#:~:text=Binary%20mode%20 . (Accessed: 29 May 2024)

N.A (n.d).What is FTP (File Transfer Protocol) [online]. Fortinet. Available at: https://www.fortinet.com/resources/cyberglossary/file-transfer-protocol-ftp-meaning#:~:text=While%20transferring%20files%2C%20FTP%20uses,block%2C%20stream%2C%20and%20compressed. (Accessed: 30 May 2024)

N.A (n.d).What is FTP server? [online]. Solarwinds. Available at: https://www.solarwinds.com/resources/it-glossary/ftp-server . (Accessed: 30 May 2024)

N.A (n.d).What is a cyclic redundancy check (CRC) in networking?[online]. Purestorage. Available at:

https://www.purestorage.com/knowledge/cyclic-redundancy-check.html#:~:text=A%20cyclic%20redundancy%20check%20(CRC)%20is%20a%20mathematical%20technique%20that,checksum%2C%20to%20the%20original%20information. (Accessed: 30 May 2024)

N.A (n.d.). Introduction to Linux Uptime. [online]. site24x7. Available at: https://www.site24x7.com/learn/linux/uptime.html (Accessed: 27 May 2024)

Postel J., Reynolds J. (1985). File Transfer Protocol (FTP). [doc]. Datatracker. Available at: https://datatracker.ietf.org/doc/html/rfc959 (Accessed 31 May 2024)

Tatsuhiko M. (2024). App:cpanminus. [online]. metacpan. Available at: https://metacpan.org/pod/App::cpanminus (Accessed: 27 May 2024)

Vikas_jk (2022).What is FTP server and how to use it. [online]. DevelopingDaily Available at:https://developingdaily.com/article/technology/what-is-ftp-server-and-how-to-use-it/249#google_vignette. (Accessed: 30 May 2024)

Villanueva J.C (2024).Active vs passive FTP simplified [online]. Jscape. Available at: https://www.jscape.com/blog/active-v-s-passive-ftp-simplified. (Accessed: 30 May 2024)

Yildirim A. (2019). TCP 3-way handshake. [online]. Medium. Available at: https://medium.com/@yildirimabdrhm/tcp-3-way-handshake-2e4d4d674ff6 (Accessed: 31 May 2024)