National University of Computer and Emerging Sciences

Laboratory Manuals

for

Computer Networks - Lab

(CL -3001)

Course Instructor	Sir Nauman Moazzam Hayat
Lab Instructor(s)	Mr. Usama Khan Mr. Muzammil Muneer
Section	BCS-6B
Semester	Spring 2023

Department of Computer Science
FAST-NU, Lahore, Pakistan

15th April, 2023

Lab Manual 07

Objective:

- Analyzing the **FTP** packets using Wireshark
- TCP Socket Programming using Multithreaded Server to handle multiple clients at the same time

Lab Statement 1: Capturing FTP packets using Wireshark (10)

Step 1: **Start a Wireshark capture.**

a. Close all unnecessary network traffic, such as the web browser, to limit the amount traffic during the Wireshark capture.

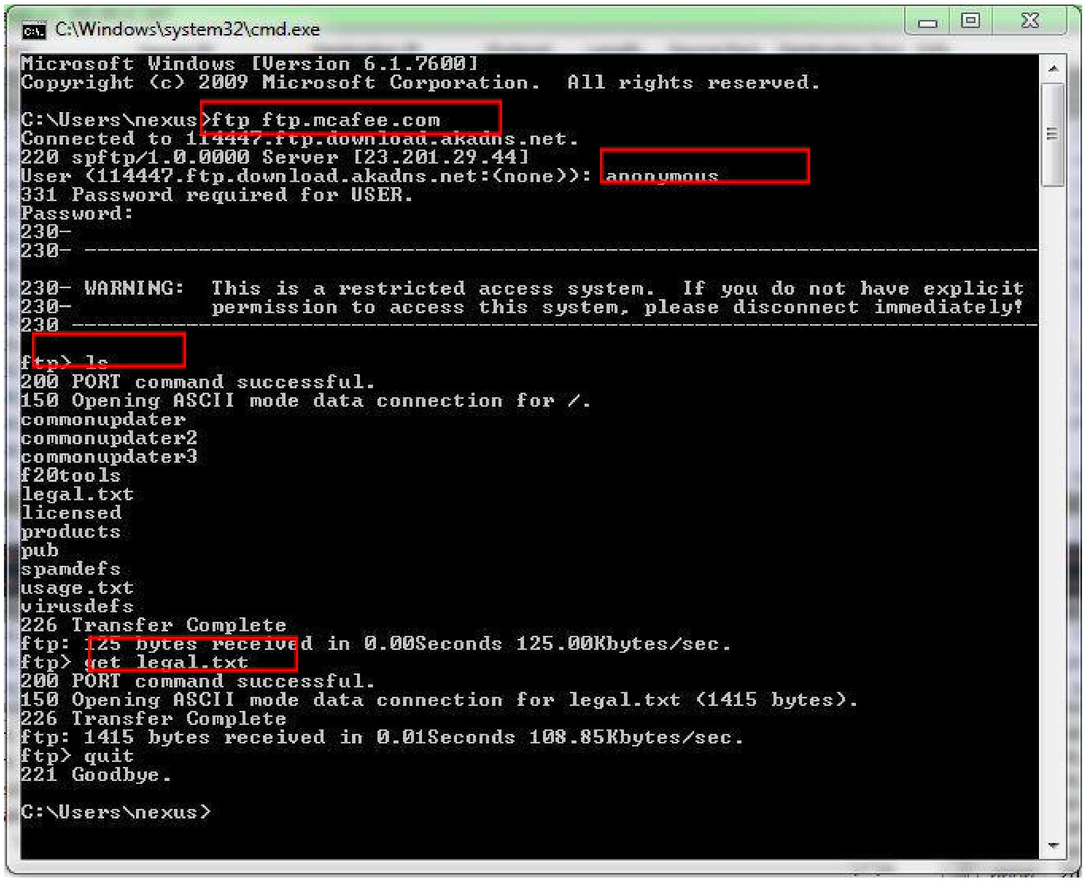
b. Start the Wireshark capture.

Step 2: **Download the .txt file.**

a. From the command prompt, enter ftp ftp.mcafee.com

b. Log into the FTP site for mcafee.com with user **anonymous** and no password.

c. Locate and download any .txt file.



C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\nexus>ftp ftp.mcafee.com
Connected to 114447.ftp.download.akadns.net.
220 spftp/1.0.0000 Server [23.201.29.44]
User (114447.ftp.download.akadns.net:(none)): anonymous
331 Password required for USER.
Password:
230-
230- -----------------------------------------------------------------------------

230- WARNING:  This is a restricted access system.  If you do not have explicit
230-           permission to access this system, please disconnect immediately!
230 -----------------------------------------------------------------------------

ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /.
commonupdater
commonupdater2
commonupdater3
f20tools
legal.txt
licensed
products
pub
spamdefs
usage.txt
virusdefs
226 Transfer Complete
ftp: 125 bytes received in 0.00Seconds 125.00Kbytes/sec.
ftp> get legal.txt
200 PORT command successful.
150 Opening ASCII mode data connection for legal.txt (1415 bytes).
226 Transfer Complete
ftp: 1415 bytes received in 0.01Seconds 108.85Kbytes/sec.
ftp> quit
221 Goodbye.

C:\Users\nexus>
```

<u>Step 3:</u> **Stop the Wireshark capture.**

<u>Step 4:</u> **View the Wireshark Main Window**

Wireshark captured many packets during the FTP session to ftp.mcafee.com. To limit the amount of data for analysis, type **tcp and ip.addr == 195.89.6.167** in the Filter. The IP address, **195.89.6.167**, is the address for ftp.mcafee.com.

<u>Step 5:</u> **Analyze the packets**

Carefully analyze the packets in Wireshark windows and answer the following question:

Use the FTP_Session.pcapng (Wireshark Capture File) to answer the questions below

1. FTP uses two port numbers: 20 and 21. Apply **tcp.port==20** and **tcp.port==21**. Analyze the result and write down the purposes of these two ports for FTP.

2. Filter out each packet using either FTP or FTP-DATA Protocol (using **ftp || ftp-data** filter). Mention each packet number and its purpose with reference to request made and response received in the above mentioned FTP Session in command line to get file legal.txt (screenshot show above). Also look for **Response Code** and **Response Arg** in the FTP Header for each packet

   (There are **19 such packets** and you have to write one/two lines explanation for each packet, what the packet is doing w.r.t FTP Session (Screenshot shown above) **e.g., Packet 104: Client asks server to send the data on IP:192.168.1.2 and Port:16341** [63(0x3F),213(0xD5) and **(0x3FD5=16341)**] **)**

Lab Statement 2: Multithreaded ECHO server using TCP (10)

You are required to design a **Multi-Threaded Echo Server and a Simple client**. The server uses a <u>TCP protocol</u> to connect to clients. Server will be listening for clients to connect to it and as soon as a client connects, it assigns a separate thread for further processing. The thread will be responsible to receive the data from the client and <u>*echo*</u> it to the client until the client sends the "DISCONNECT" command. **The server can handle maximum 3 clients at a time.**

Client will be a simple program which after connecting to the server will take the input from the user and send it to the server, then outputs the response on the terminal received from the server. It will do the same until user enters “DISCONNECT”. Upon entering “DISCONNECT” the client shall close the socket and exit.

Following are the steps which Server should perform:

1. Receives a connection request from client and pass the socket descriptor returned by the accept() to the thread and goes back to listen for more connections for **clients< 4**. If the fourth client tries to connect then server sends the client message that “**Server Full**”

2. Meanwhile this is what the thread do
   - Receive what the client sends.
   - Echo back what client sends.
   - If client has sent “DISCONNECT” then close the socket and quit.

3. Receive more data from the same client.

You should cater 3multiple clients that will be sending connection requests to server.

Following are the steps which Clients should perform:

1. Take input from the user.
2. Send input to the server
3. If input is “DISCONNECT” then close the socket and exit otherwise continue to step 4.
4. Wait for server’s response.
5. Print the server’s repose.
6. Go back to step 1.

Very important:

Make sure you add the proper header file **<pthread.h>** to make use of threading in your C language code. At the compilation time, you must enter **gcc –pthread server.c -o server.out.** Otherwise your code will give errors.

Pthread function syntax:

- **int pthread_create(pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);**

1. `pthread_t *thread`: the actual thread object that contains pthread id
2. `pthread_attr_t *attr`: attributes to apply to this thread(use NULL for default attributes)
3. `void *(*start_routine)(void *)`: the function this thread executes
4. `void *arg`: arguments to pass to thread function above

- **`voidpthread_exit(void*value_ptr)`**

  `pthread_exit()` terminates the thread and provides the pointer `*value_ptr` available to any `pthread_join()` call.

- **`intpthread_join(pthread_t thread, void**value_ptr);`**

  `pthread_join()` suspends the calling thread to wait for successful termination of the thread specified as the first argument `pthread_t thread` with an optional `*value_ptr` data passed from the terminating thread's call to `pthread_exit()`.