Assignment 6

Socket Programming

By Ziming Song

[zs2815@nyu.edu](mailto:zs2815@nyu.edu)

# **Part 1: netcat TCP chat**

## **implement a TCP chat over a network.** **Let R2 be the server and KALI the client.**

In R2’s terminal, type “hi I am R2”

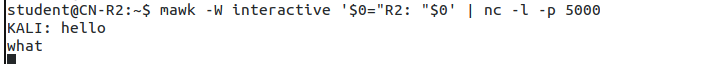


In Kali’s terminal, we can see “hi I am R2”

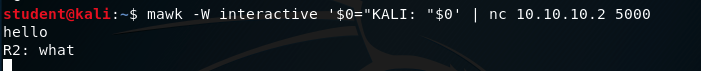


## **Screenshots of R2 and KALI showing the netcat TCP chat [10 points].**

In Kali’s terminal, type “hello”. Then in R2’s terminal, type “what”



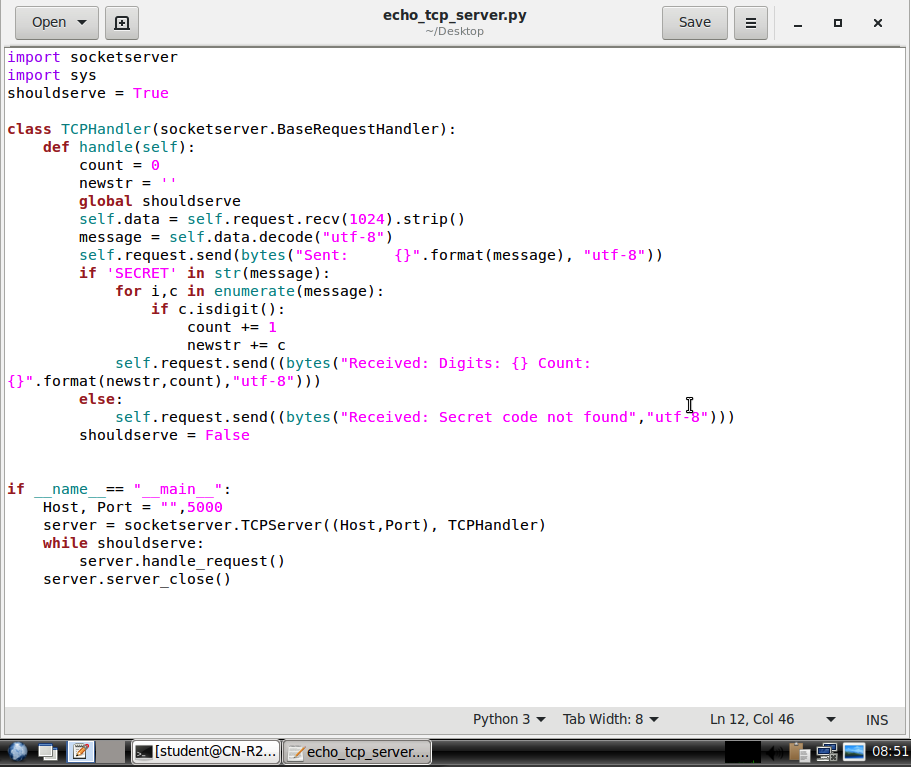
Screenshot of R2



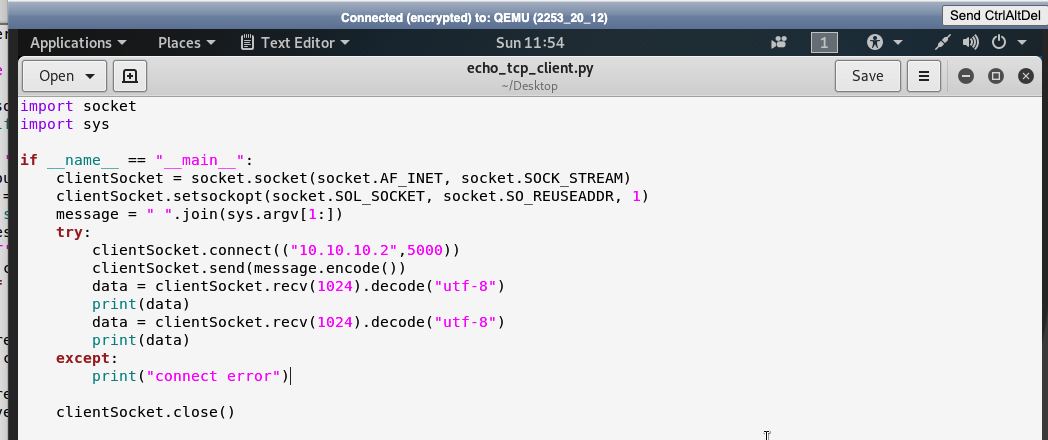
Screenshot of Kali

# **Part 2: Client-server with secret code**

## **Screenshots of echo\_tcp\_server.py and echo\_tcp\_client.py(showing all code) [10 points]**

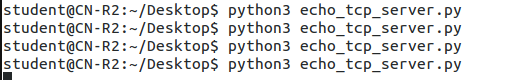


Screenshot of echo\_tcp\_server.py on R2

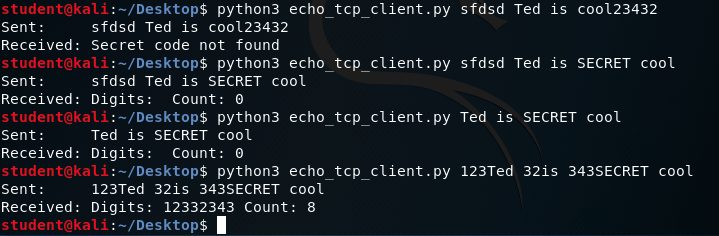


Screenshot of echo\_tcp\_client.py on Kali

1. **Screenshots Screenshots showing the behavior of Part 2. Make sure to include cases with and without the secret code.**

****

Screenshot of input on R2

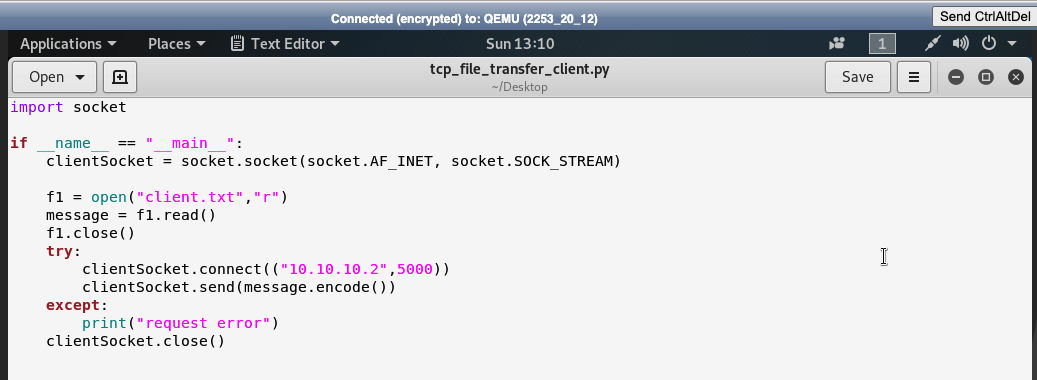


Screenshot of result & behavior on Kali

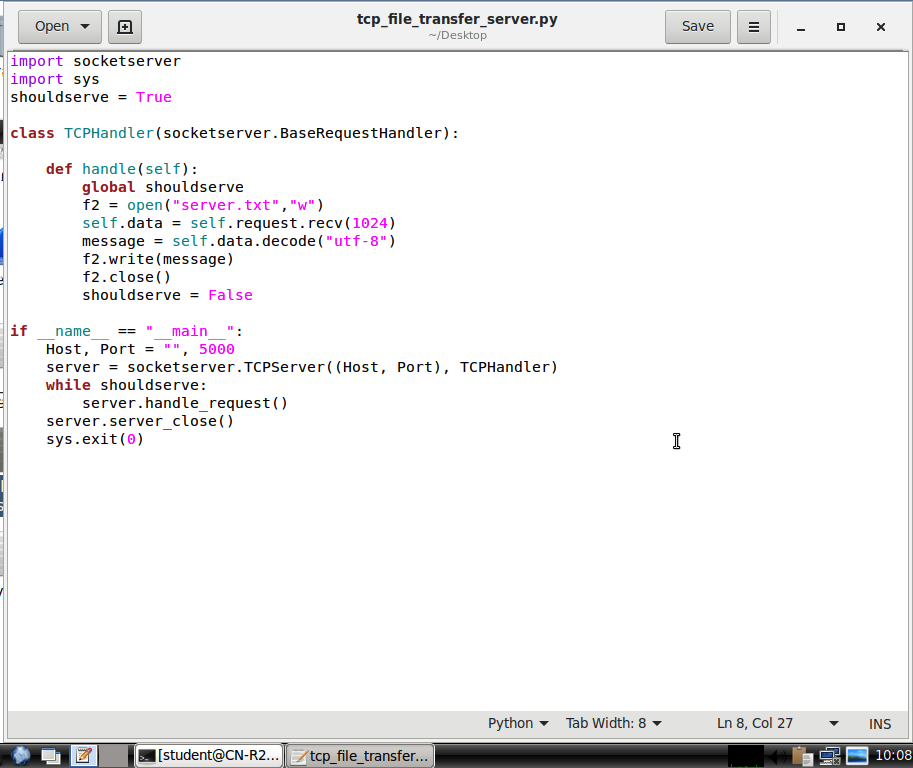
# **Part 3: Client-server with secret code**

## **Screenshots of tcp\_file\_transfer\_server.py and tcp\_file\_transfer\_client.py**

## **(showing all code) [10 points]**

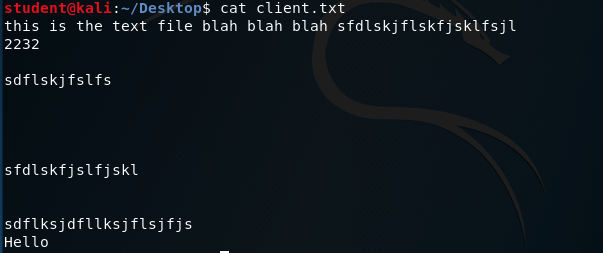


Screenshot of tcp\_file\_transfer\_client.py on Kali

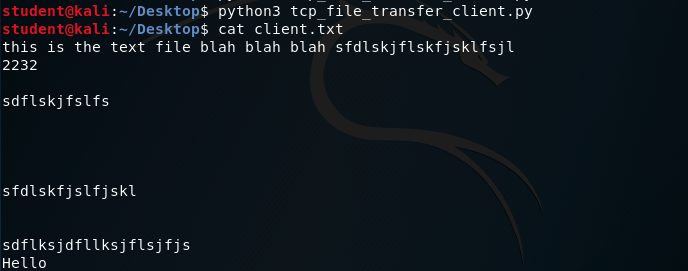


Screenshot of tcp\_file\_transfer\_server.py on R2

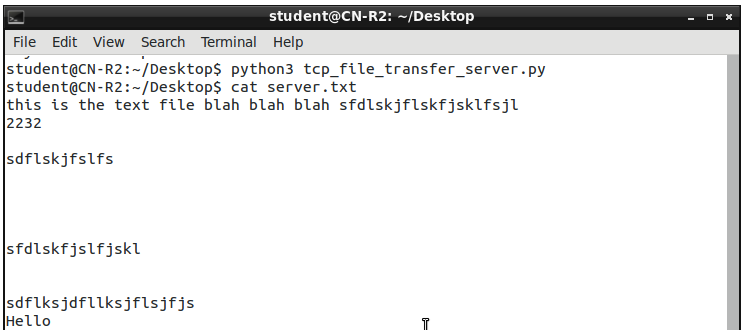
## **Screenshots showing the file transfer in Part 3: show the original file on KALI, the KALI terminal after transferring, and the transferred file on R2. [5 points]**



Screenshot of the original file “client.txt” on KALI



Screenshot of KALI terminal and “client.txt” after transferring

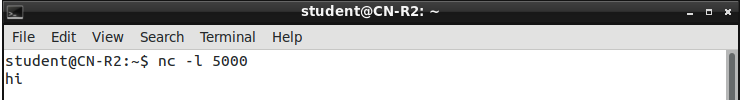


Screenshot of the transferred file “server.txt” on R2

# **Part 4: Questions**

## **In netcat, you specified the port on which the server should listen but did not specify the port the server should use to send a message to the client. Which client port does your netcat server send to? Use Wireshark to answer the question and include a screenshot. [10 points]**

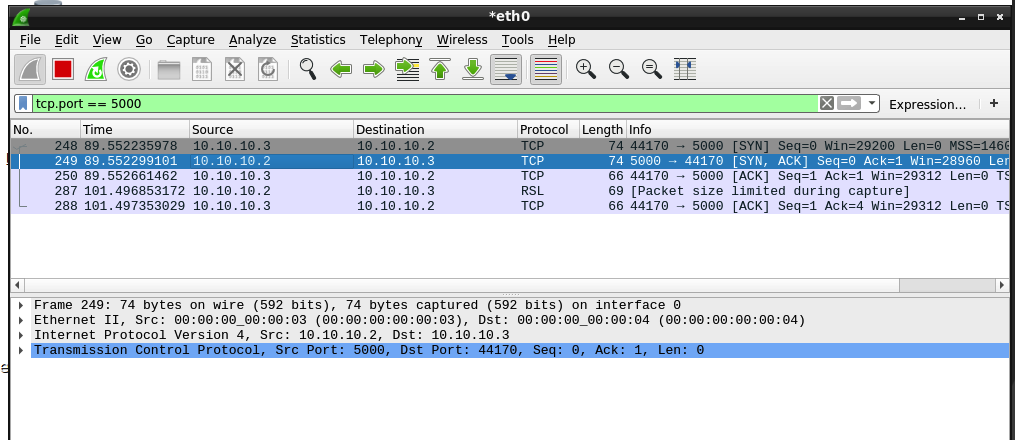
After netcat, R2 send a message hello to Kali.



Screenshot of R2 terminal



Screenshot of Kali terminal



Screenshot of wireshark on R2

In screenshot of wireshark, we can see client port is 44170.

## **Briefly explain your code from Part 2 and Part 3. In your explanation, focus not on the syntax but on the TCP communication establishment and flow. [10 points]**

In Part2 echo\_tcp\_server.py:

This code defines a basic TCP server using the socketserver module. The server listens on a specific port (5000). The TCPHandler class handles incoming connections. Upon receiving data, it decodes it and checks if the word 'SECRET' is present. If found, it sends a response containing the original message. It also counts and extracts digits from the message, sending a response with the digit count and extracted digits. If 'SECRET' is not found, it sends a response indicating that the secret code was not found. The server runs in a loop until the global variable shouldserve is set to False.

In Part2 echo\_tcp\_client.py:

This code establishes a TCP client socket for communication. It attempts to connect to a server at the address (R2’s IP address) and port (5000). The client sends a message formed by joining command-line arguments. If the connection is successful, it receives and prints two sets of data from the server using UTF-8 encoding. If an exception occurs during the connection or data reception, it prints a "connect error" message. Finally, the client socket is closed.

In Part3 tcp\_file\_transfer\_server.py:

This code sets up a basic TCP server using the socketserver module. The server listens on a specified port (5000) for incoming connections. When a connection is established, it handles the communication in the TCPHandler class. The server continuously listens for incoming data, receives messages (assuming newline-separated messages), and writes them to a file ("server.txt"). The server runs in a loop until a condition (shouldserve) is met, and then it gracefully shuts down. The server's main loop uses handle\_request() to process incoming requests, and server\_close() is called to release the port when the loop exits.

In Part3 tcp\_file\_transfer\_client.py:

This code establishes a TCP connection to a server at IP address 10.10.10.2(R2’s IP address) and port 5000. It reads the contents of a file named "client.txt" and sends the data over the established connection. If the connection or sending process encounters an exception, it prints a "request error" message. Finally, it closes the TCP connection. This script essentially acts as a basic TCP client, initiating communication with a server and transmitting data from a file.

## **What does the socket system call return? [5 points]**

The socket system call in Python returns a new socket object.

## **What does the bind system call return? Who calls bind (client/server)? [5 points]**

The bind system call in Python returns “None” upon success and raises an exception if an error occurs. The server typically calls bind to specify the address and port it will be reachable at. Clients generally do not call bind since the operating system assigns a local port to them when connecting to a server.

## **Suppose you wanted to send an urgent message from a remote client to a server as fast as possible. Would you use UDP or TCP? Why? (Hint: compare RTTs.) [10 points]**

UDP.

UDP is a connectionless protocol with lower overhead and faster transmission, making it ideal for scenarios where low latency is crucial. Unlike TCP, UDP does not establish a connection before sending data and does not perform extensive error-checking and retransmission of lost packets, reducing the round-trip time (RTT). So in this situation, where minimizing latency is a priority, UDP provides a more efficient and quicker communication method.

## **What is Nagle’s algorithm? What problem does it aim to solve and how? [10 points]**

Nagle's algorithm is a technique used in TCP to improve the efficiency of small packet transmissions. The problem it aims to solve "small packet problem," where the transmission of many small messages in quick succession can lead to inefficient use of network resources.

Nagle's algorithm works by delaying the transmission of small packets and combining them into larger, more efficient packets. It introduces a small delay (typically around 200 milliseconds) before sending small packets. If additional data arrives during this delay, the data is grouped together, reducing the overhead associated with sending multiple small packets separately. This helps optimize network utilization and enhances overall performance, particularly in scenarios involving frequent small data transfers.

## **Explain one potential scenario in which delayed ACK could be problematic. [10 points]**

In interactive communication like video chat, immediate feedback is crucial for a smooth user experience. If the receiver delays sending ACKs, it can introduce unnecessary latency. The sender might be waiting for acknowledgments before sending additional data, causing a slowdown in the communication flow. For example, video and voice can become laggy.