Assignment 6

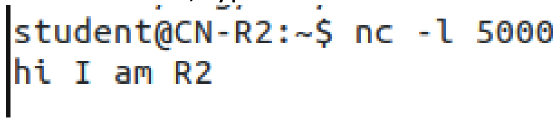
Socket Programming

By Ziming Song
zs2815@nyu.edu

Part 1: netcat TCP chat

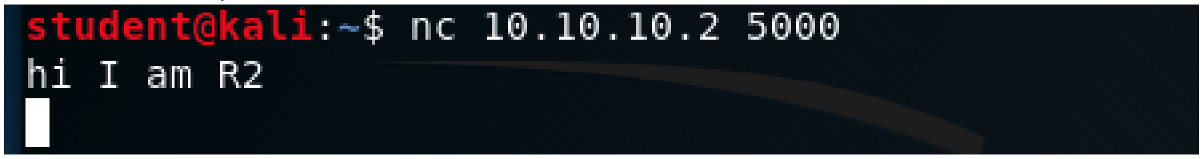
a. **implement a TCP chat over a network. Let R2 be the server and KALI the client.**

In R2's terminal, type "hi I am R2"



```
student@CN-R2:~$ nc -l 5000
hi I am R2
```

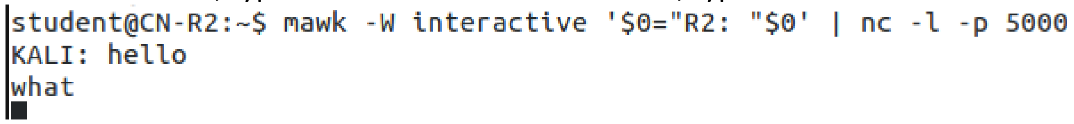
In Kali's terminal, we can see "hi I am R2"



```
student@kali:~$ nc 10.10.10.2 5000
hi I am R2
```

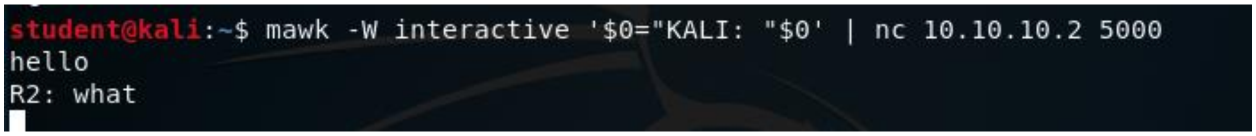
b. **Screenshots of R2 and KALI showing the netcat TCP chat [10 points].**

In Kali's terminal, type "hello". Then in R2's terminal, type "what"



```
student@CN-R2:~$ mawk -W interactive '$0="R2: "$0' | nc -l -p 5000
KALI: hello
what
```

Screenshot of R2

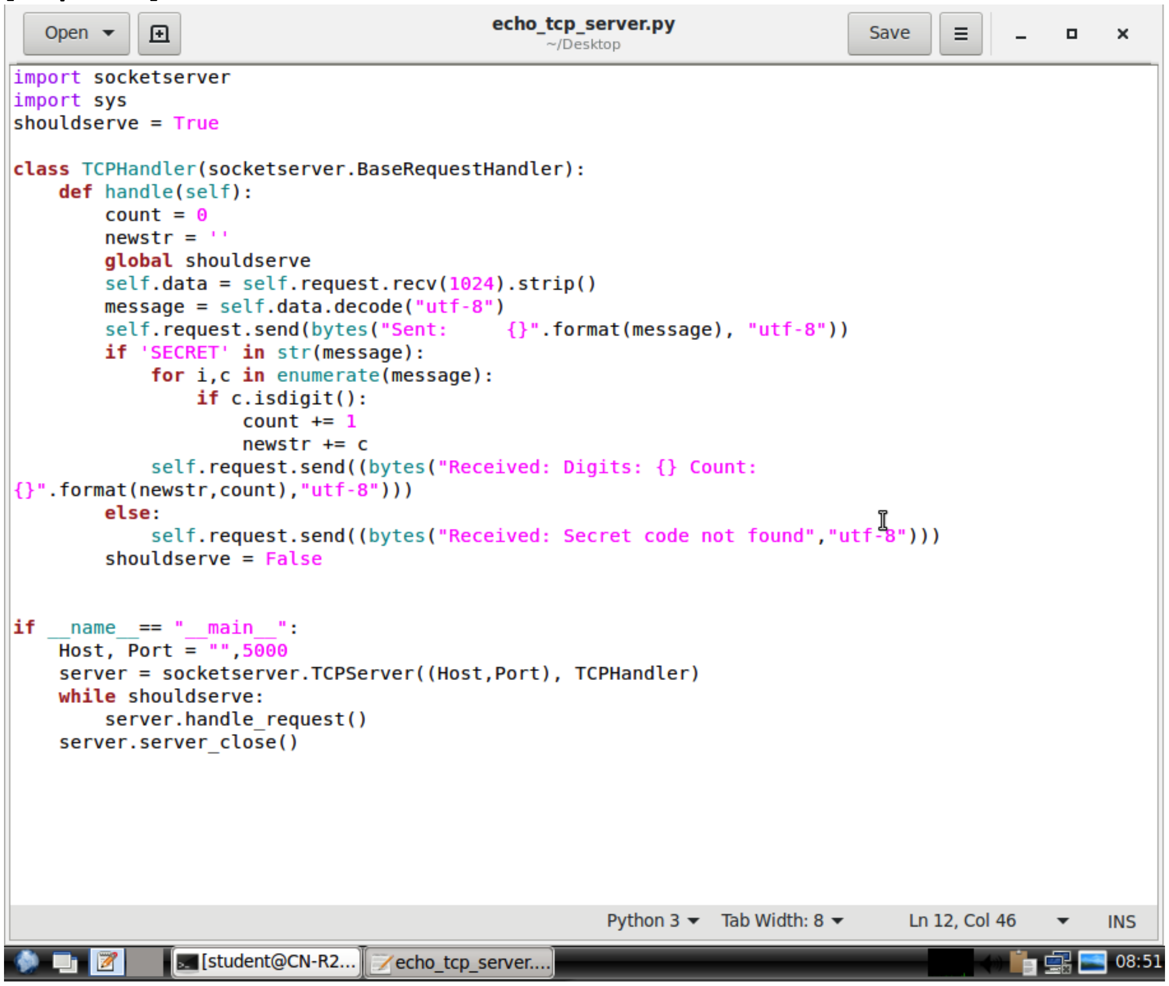


```
student@kali:~$ mawk -W interactive '$0="KALI: "$0' | nc 10.10.10.2 5000
hello
R2: what
```

Screenshot of Kali

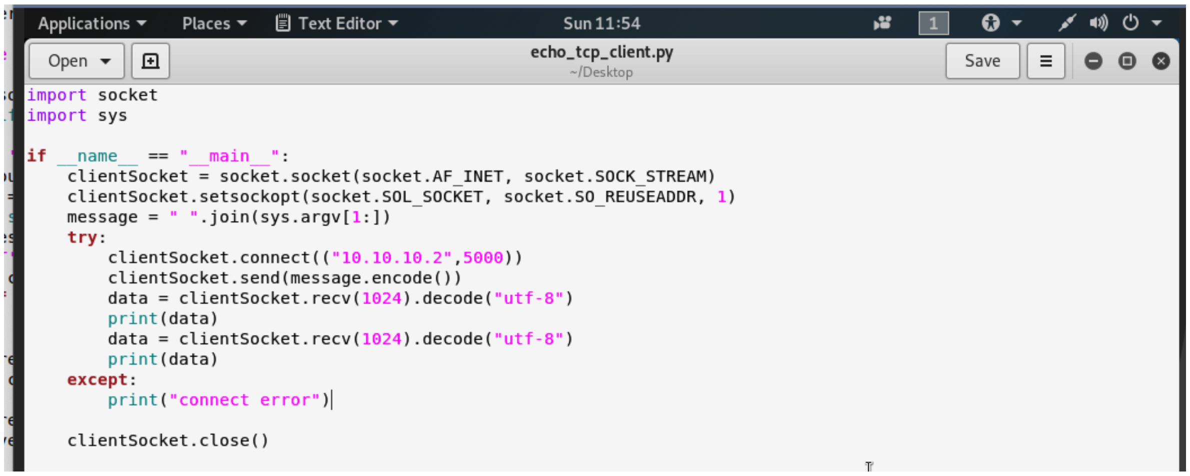
Part 2: Client-server with secret code

a. **Screenshots of echo_tcp_server.py and echo_tcp_client.py(showing all code) [10 points]**



```python
import socketserver
import sys
shouldserve = True

class TCPHandler(socketserver.BaseRequestHandler):
    def handle(self):
        count = 0
        newstr = ''
        global shouldserve
        self.data = self.request.recv(1024).strip()
        message = self.data.decode("utf-8")
        self.request.send(bytes("Sent:     {}".format(message), "utf-8"))
        if 'SECRET' in str(message):
            for i,c in enumerate(message):
                if c.isdigit():
                    count += 1
                    newstr += c
            self.request.send((bytes("Received: Digits: {} Count:
{}".format(newstr,count),"utf-8")))
        else:
            self.request.send((bytes("Received: Secret code not found","utf-8")))
        shouldserve = False


if __name__== "__main__":
    Host, Port = "",5000
    server = socketserver.TCPServer((Host,Port), TCPHandler)
    while shouldserve:
        server.handle_request()
    server.server_close()
```

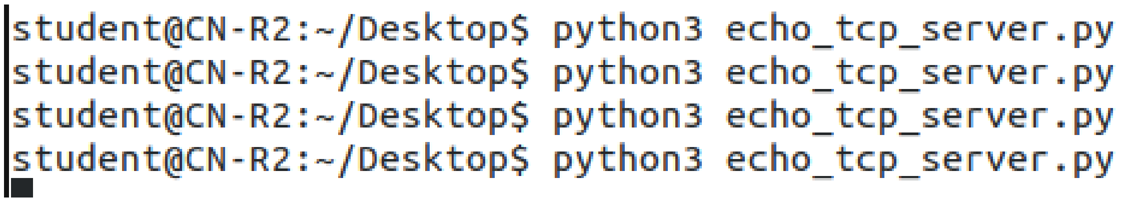
Screenshot of echo_tcp_server.py on R2



```python
import socket
import sys

if __name__ == "__main__":
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    clientSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    message = " ".join(sys.argv[1:])
    try:
        clientSocket.connect(("10.10.10.2",5000))
        clientSocket.send(message.encode())
        data = clientSocket.recv(1024).decode("utf-8")
        print(data)
        data = clientSocket.recv(1024).decode("utf-8")
        print(data)
    except:
        print("connect error")

    clientSocket.close()
```

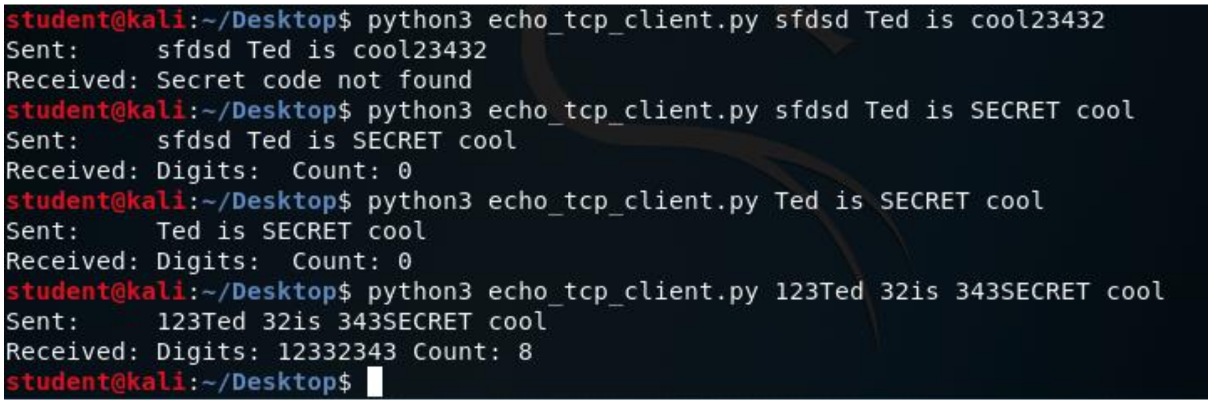
Screenshot of echo_tcp_client.py on Kali

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



```
student@CN-R2:~/Desktop$ python3 echo_tcp_server.py
student@CN-R2:~/Desktop$ python3 echo_tcp_server.py
student@CN-R2:~/Desktop$ python3 echo_tcp_server.py
student@CN-R2:~/Desktop$ python3 echo_tcp_server.py
```

Screenshot of input on R2

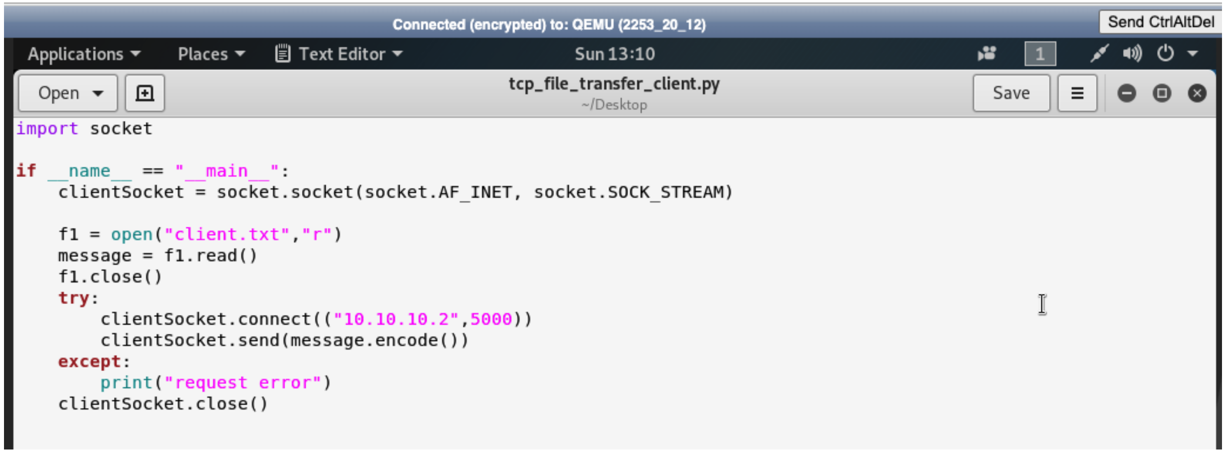


```
student@kali:~/Desktop$ python3 echo_tcp_client.py sfdsd Ted is cool23432
Sent:     sfdsd Ted is cool23432
Received: Secret code not found
student@kali:~/Desktop$ python3 echo_tcp_client.py sfdsd Ted is SECRET cool
Sent:     sfdsd Ted is SECRET cool
Received: Digits:  Count: 0
student@kali:~/Desktop$ python3 echo_tcp_client.py Ted is SECRET cool
Sent:     Ted is SECRET cool
Received: Digits:  Count: 0
student@kali:~/Desktop$ python3 echo_tcp_client.py 123Ted 32is 343SECRET cool
Sent:     123Ted 32is 343SECRET cool
Received: Digits: 12332343 Count: 8
student@kali:~/Desktop$
```

Screenshot of result & behavior on Kali

Part 3: Client-server with secret code

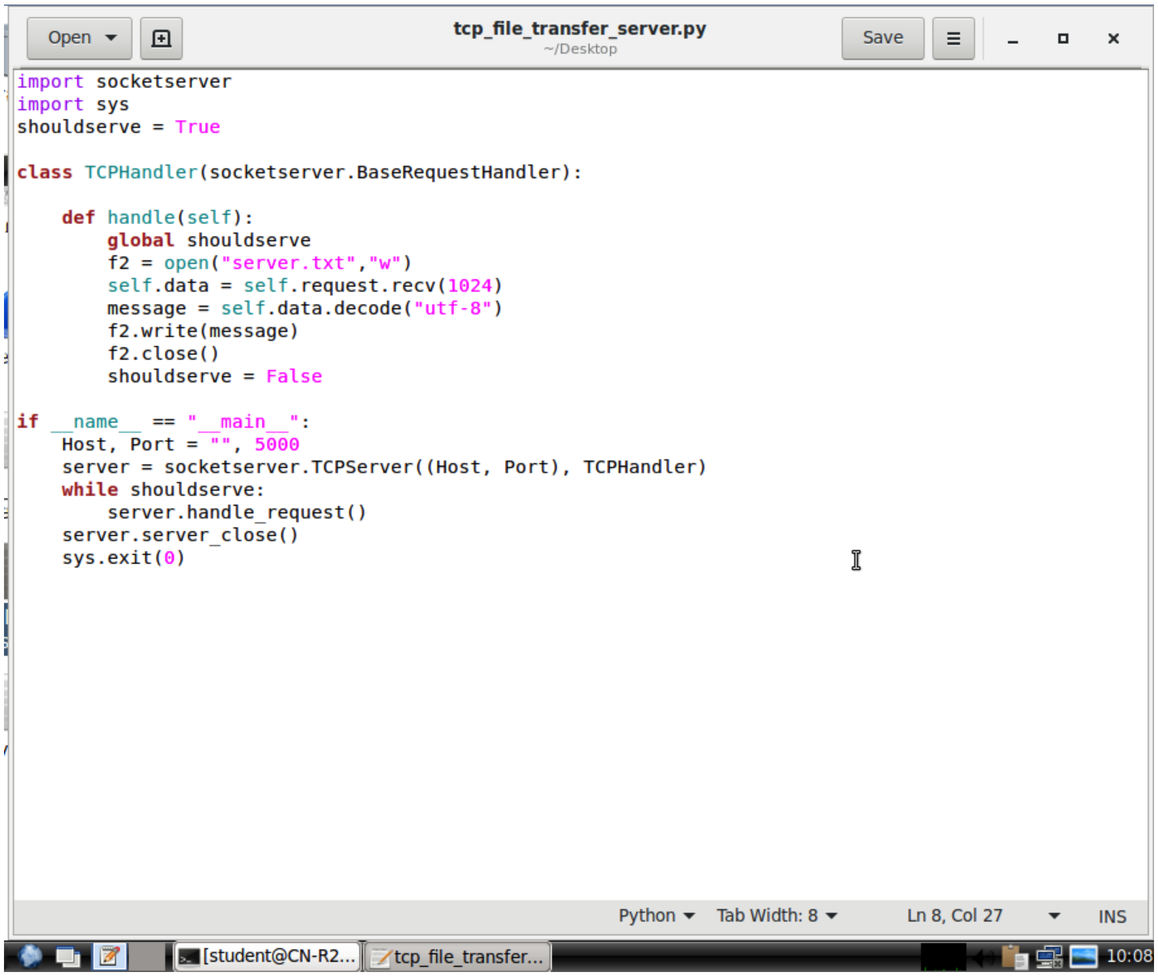
a. **Screenshots of tcp_file_transfer_server.py and tcp_file_transfer_client.py (showing all code) [10 points]**



Connected (encrypted) to: QEMU (2253_20_12)
Send CtrlAltDel
Applications
Places
Text Editor
Sun 13:10
Open
tcp_file_transfer_client.py
~/Desktop
Save

```
import socket

if __name__ == "__main__":
    clientSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    f1 = open("client.txt","r")
    message = f1.read()
    f1.close()
    try:
        clientSocket.connect(("10.10.10.2",5000))
        clientSocket.send(message.encode())
    except:
        print("request error")
    clientSocket.close()
```

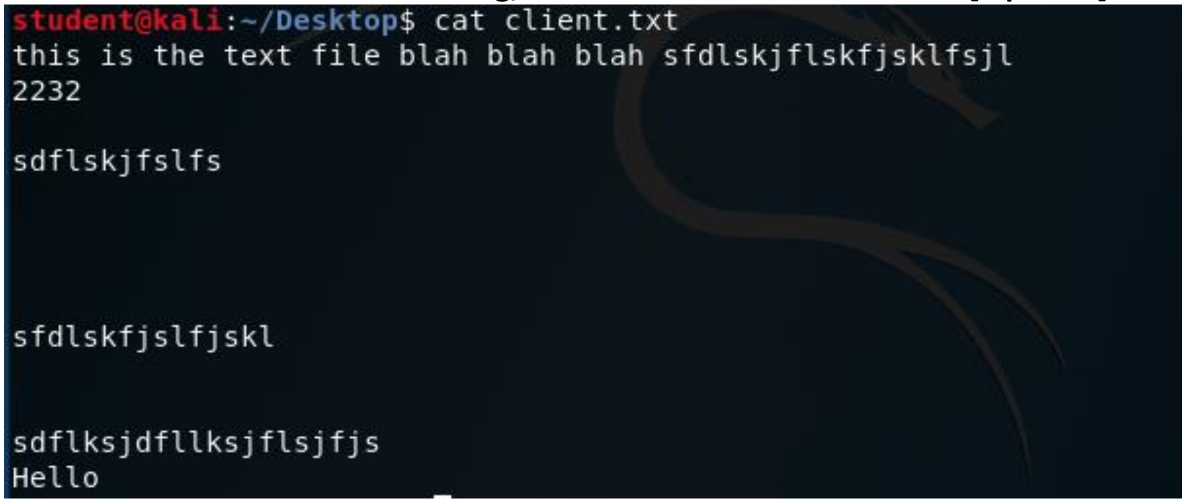
Screenshot of tcp_file_transfer_client.py on Kali



```
import socketserver
import sys
shouldserve = True

class TCPHandler(socketserver.BaseRequestHandler):

    def handle(self):
        global shouldserve
        f2 = open("server.txt","w")
        self.data = self.request.recv(1024)
        message = self.data.decode("utf-8")
        f2.write(message)
        f2.close()
        shouldserve = False

if __name__ == "__main__":
    Host, Port = "", 5000
    server = socketserver.TCPServer((Host, Port), TCPHandler)
    while shouldserve:
        server.handle_request()
    server.server_close()
    sys.exit(0)
```

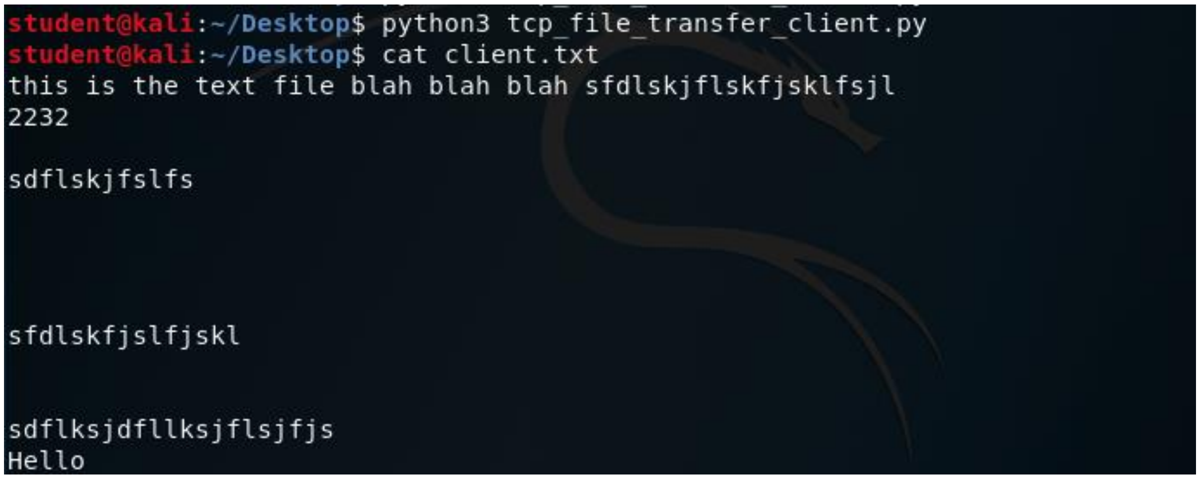
Screenshot of tcp_file_transfer_server.py on R2

b. 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]



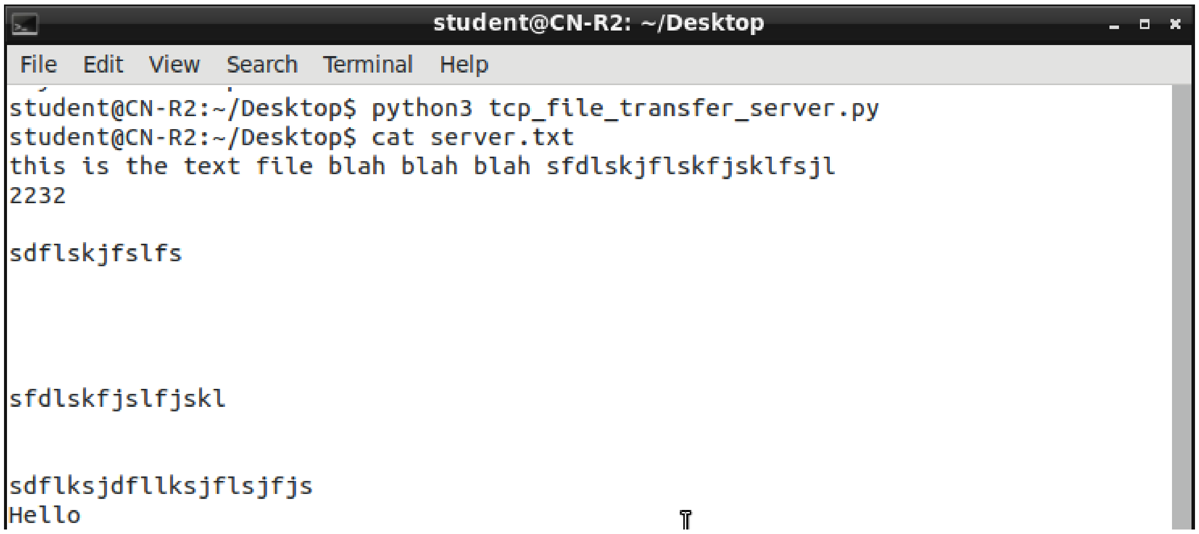
```
student@kali:~/Desktop$ cat client.txt
this is the text file blah blah blah sfdlskjflskfjsklfsjl
2232

sdflskjfslfs




sfdlskfjslfjskl



sdflksjdfllksjflsjfjs
Hello
```

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



```
student@kali:~/Desktop$ python3 tcp_file_transfer_client.py
student@kali:~/Desktop$ cat client.txt
this is the text file blah blah blah sfdlskjflskfjsklfsjl
2232

sdflskjfslfs




sfdlskfjslfjskl


sdflksjdfllksjflsjfjs
Hello
```

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



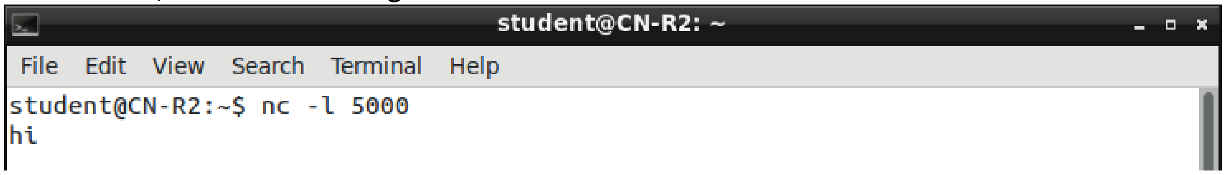
```
student@CN-R2:~/Desktop$ python3 tcp_file_transfer_server.py
student@CN-R2:~/Desktop$ cat server.txt
this is the text file blah blah blah sfdlskjflskfjsklfsjl
2232

sdflskjfslfs




sfdlskfjslfjskl


sdflksjdfllksjflsjfjs
Hello
```

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

Part 4: Questions

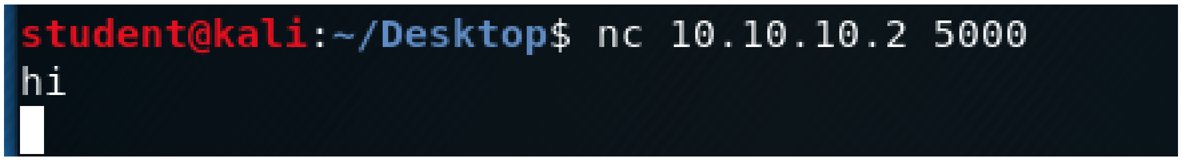
a. **In netcat, you specified the port on which the server should listen but did not specify the port the server should use to send a message to the client. Which client port does your netcat server send to? Use Wireshark to answer the question and include a screenshot. [10 points]**

After netcat, R2 send a message hello to Kali.



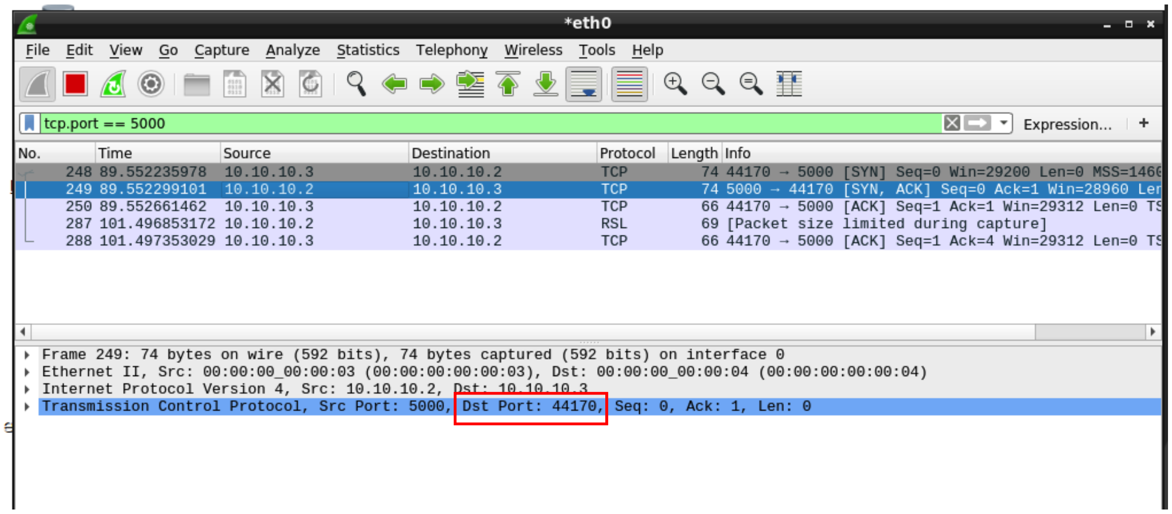
```
student@CN-R2:~$ nc -l 5000
hi
```

Screenshot of R2 terminal



```
student@kali:~/Desktop$ nc 10.10.10.2 5000
hi
```

Screenshot of Kali terminal


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...

```
tcp.port == 5000

No.   Time           Source      Destination  Protocol Length Info
  248 89.552235978   10.10.10.3  10.10.10.2   TCP      74 44170 → 5000 [SYN] Seq=0 Win=29200 Len=0 MSS=146
  249 89.552299101   10.10.10.2  10.10.10.3   TCP      74 5000 → 44170 [SYN, ACK] Seq=0 Ack=1 Win=28960 Le
  250 89.552661462   10.10.10.3  10.10.10.2   TCP      66 44170 → 5000 [ACK] Seq=1 Ack=1 Win=29312 Len=0 T
  287 101.496853172  10.10.10.2  10.10.10.3   RSL      69 [Packet size limited during capture]
  288 101.497353029  10.10.10.3  10.10.10.2   TCP      66 44170 → 5000 [ACK] Seq=1 Ack=4 Win=29312 Len=0 T

Frame 249: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
Ethernet II, Src: 00:00:00_00:00:03 (00:00:00:00:00:03), Dst: 00:00:00_00:00:04 (00:00:00:00:00:04)
Internet Protocol Version 4, Src: 10.10.10.2, Dst: 10.10.10.3
Transmission Control Protocol, Src Port: 5000, Dst Port: 44170, Seq: 0, Ack: 1, Len: 0
```

Screenshot of wireshark on R2

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

b. 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.

c. **What does the socket system call return? [5 points]**
The socket system call in Python returns a new socket object.

d. **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.

e. **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.

f. **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.

g. 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.