

Socket Programming

Instructions

The goal of this lab is to learn about client-server socket programming.

There are three parts to this lab.

- In Part 1, you will use a TCP/UDP utility to set up a two-way chat over a network.
- In Parts 2 and 3, you will configure R2 as a server and Kali as a client following the guidelines explained below. Use the `socketserver` Python 3 framework for both parts. Click the link below to familiarize yourself with basic usage of the `socketserver` framework:

<https://docs.python.org/3/library/socketserver.html>

Note: you may want to write the code on your local machine and transfer it to R2 and Kali using either git or SFTP. See the VITAL User Guide for more information.

Part 1: netcat TCP chat

We will begin with `netcat`—a TCP/UDP utility which comes preinstalled on most UNIX systems. As quoted from the man page (open a terminal and type `man netcat`), "The `nc` (or `netcat`) utility is used for just about anything under the sun involving TCP or UDP. It can open TCP connections, send UDP packets, listen on arbitrary TCP and UDP ports, do port scanning, and deal with both IPv4 and IPv6."

We will use `netcat` to create a two-way chat over a network. The purpose of this section is to see an example of how easy it can be to implement a TCP chat over a network.

Let R2 be the server and KALI the client. Type the following commands:

R2 (Server):

```
nc -l <port> // use any port that is not reserved (e.g., 5000)
```

KALI (Client):

```
nc <server hostname or IP address> <port>
```

On R2, type a message in the terminal window and press enter. It should be viewable in [Kali's](#) terminal. Type a message in [Kali's](#) terminal window and press enter. It should be viewable in R2's terminal. Type a few messages on each side to make a conversation.

Press `CTRL-C` in both R2 and KALI to close the connection.

Version: 4/7/2022 1

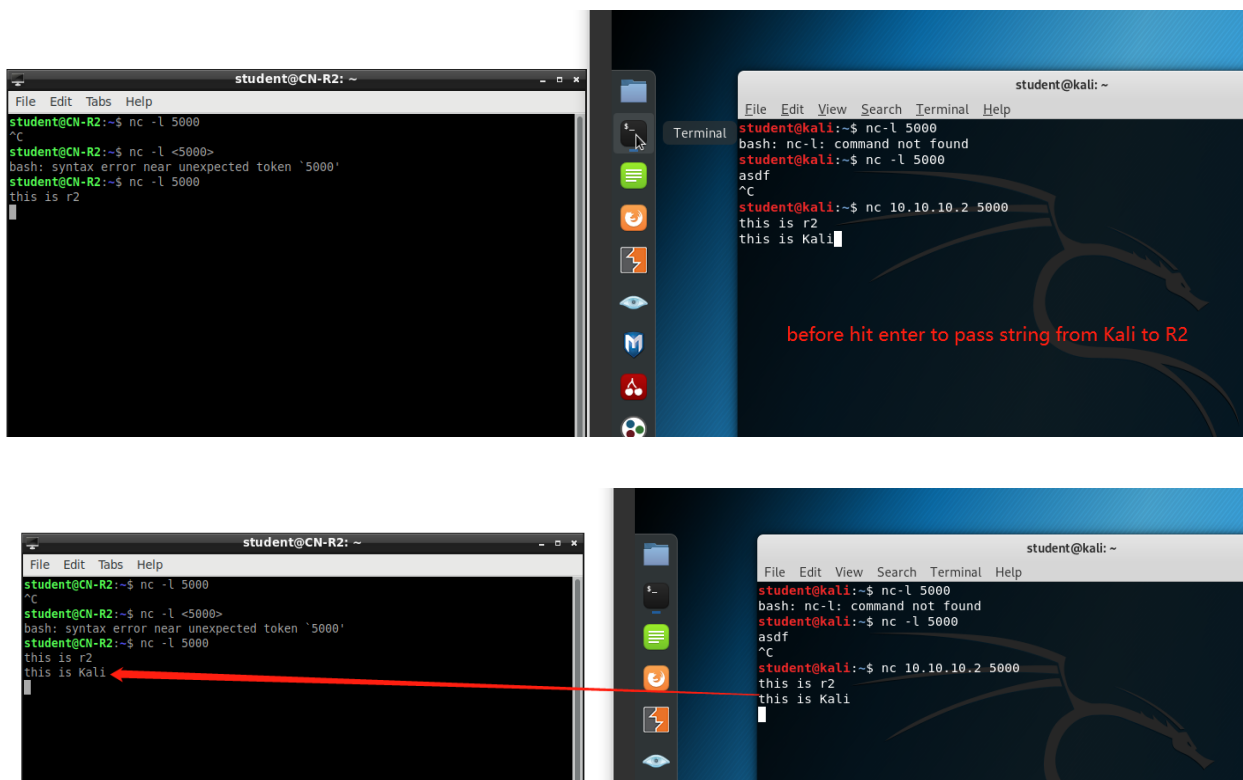
The only problem with this chat is that it is hard to tell who sent each message. Our goal is for the chat window to behave like this:

R2 types: "Hi, my name is R2"

Output in R2 and KALI terminal: "R2: Hi, my name is R2"

KALI types: "Hi R2, my name is KALI. Nice to meet you."

Output in R2 and KALI terminal: "KALI: Hi R2, my name is KALI. Nice to meet you."



Add a username ("R2" or "KALI") so that each message can be identified by its sender.

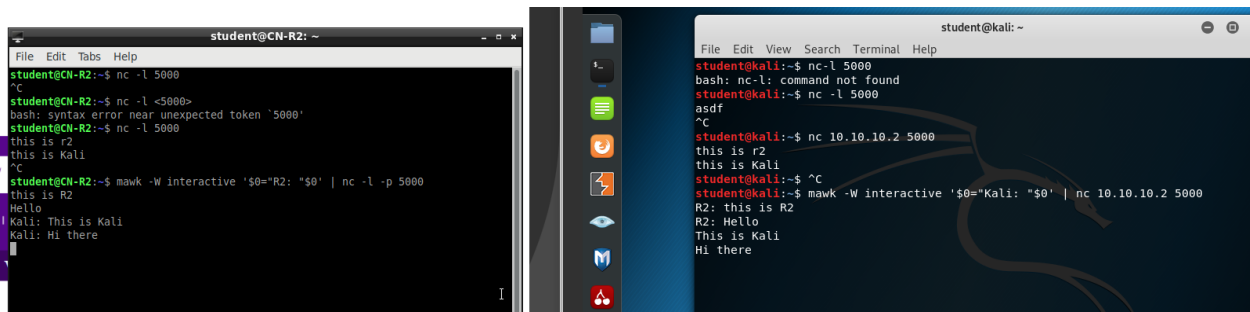
R2 (Server):

```
mawk -W interactive '$0="R2: "$0' | nc -l -p <port_number>
```

KALI (Client):

```
mawk -W interactive '$0="KALI: "$0' | nc <server IP> <port_number>
```

Create a short conversation between R2 and KALI and take a screenshot of each terminal window showing the chat.



Part 2: Client-server with secret code

You should write two files for this part: **echo_tcp_server.py** (on R2) and **echo_tcp_client.py** (on KALI). The client should send a string to the server, and the server should receive it. If the string contains the secret code "SECRET", the server should return all the digits in the string as well as the total number of digits. If the string does not contain the secret code, the server should respond with the message, "Secret code not found." The client should receive the output. The output format and behavior should match the example below. Example: client console view (CN-R4 should be CN-Kali)

```

student@CN-R4: ~/tcp
File Edit View Search Terminal Help
student@CN-R4:~/tcp$ python3 echo_tcp_client.py sfdsd Ted is cool23432
Sent:      sfdsd Ted is cool23432
Received: Secret code not found.
student@CN-R4:~/tcp$ python3 echo_tcp_client.py Ted is SECRET cool
Sent:      Ted is SECRET cool
Received: Digits: Count: 0
student@CN-R4:~/tcp$ python3 echo_tcp_client.py 123Ted 32is 343SECRET cool
Sent:      123Ted 32is 343SECRET cool
Received: Digits: 12332343 Count: 8
student@CN-R4:~/tcp$

```

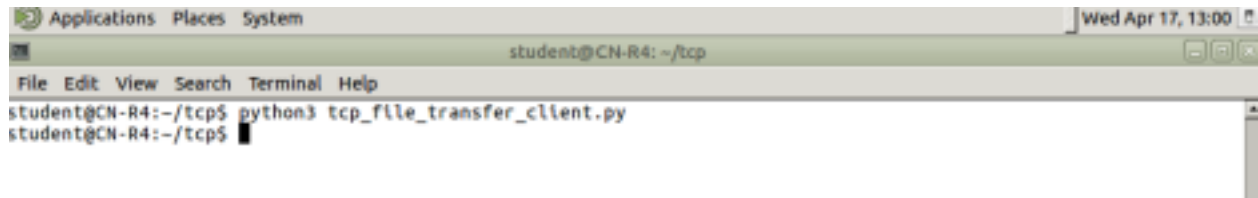
Version: [4/7/2022](#) 2

Part 3: Client-server with file transfer

You should write two files for this part: **tcp_file_transfer_server.py** (on R2) and **tcp_file_transfer_client.py** (on KALI). The client should create a file (any text file with

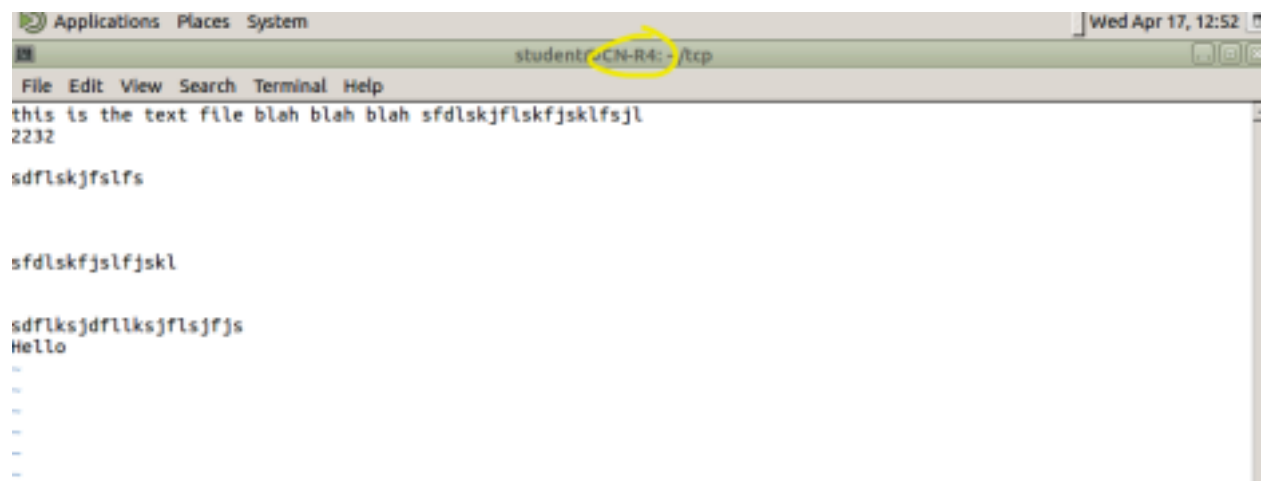
some content will suffice) and send the data in this file to the server. The server should receive the data and write it to a file. The resulting file should be exactly the same as the file on the client side. Once the transfer is complete, the connection should be closed. The output format and behavior should match the example below.

Example: client contesole view (CN-R4 should be CN-Kali)



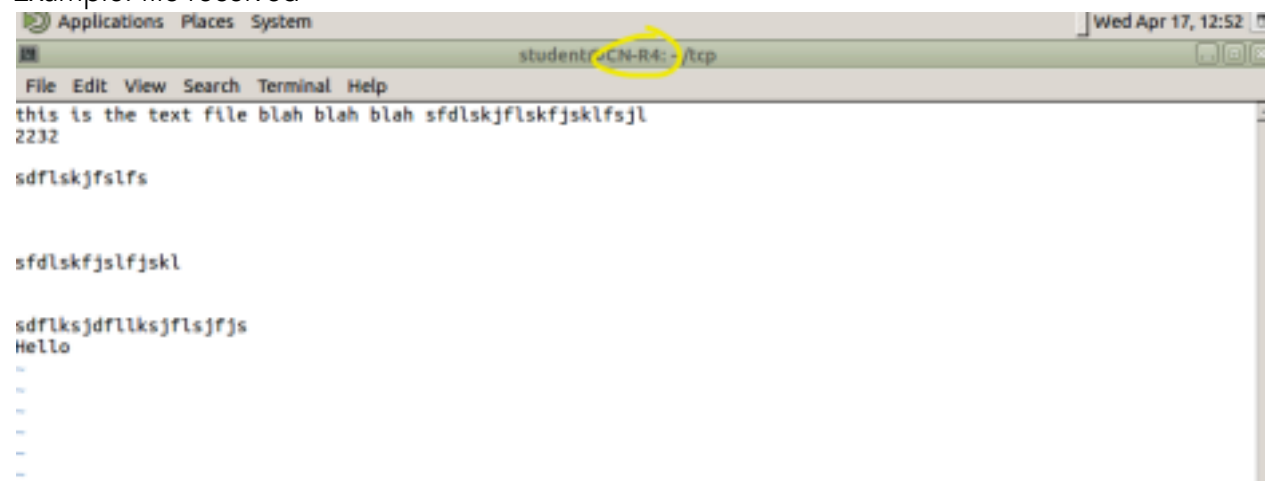
```
Applications Places System Wed Apr 17, 13:00
student@CN-R4: ~/tcp
File Edit View Search Terminal Help
student@CN-R4:~/tcp$ python3 tcp_file_transfer_client.py
student@CN-R4:~/tcp$
```

Example: file to be transferred



```
Applications Places System Wed Apr 17, 12:52
student@CN-R4: ~/tcp
File Edit View Search Terminal Help
this is the text file blah blah blah sfdlskjflskfjsklfsjl
2232
sfdlskjfslfs
sfdlskfjslfjskl
sfdlksjdfllksjflsjfjs
Hello
..
..
..
..
```

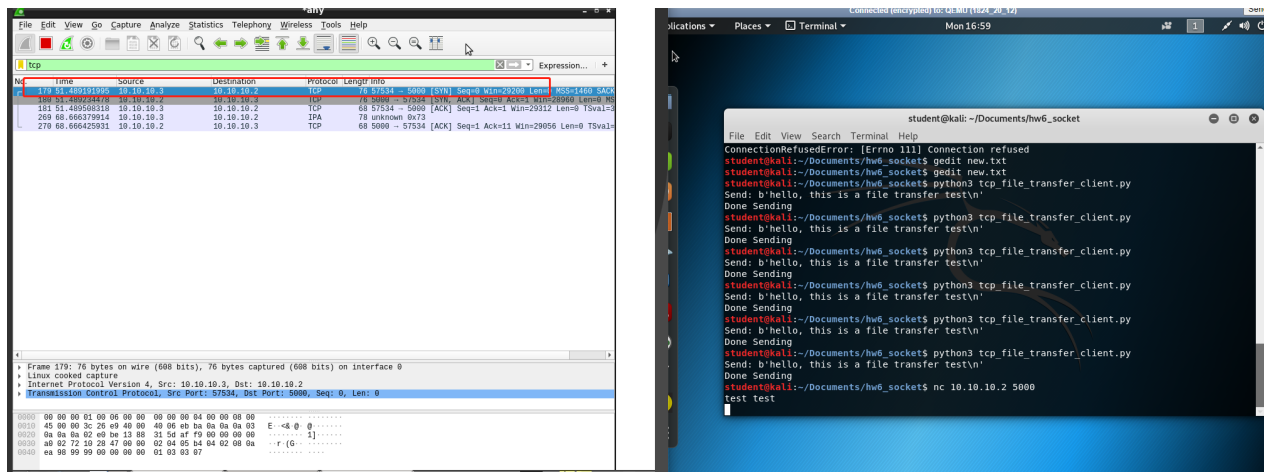
Example: file received



```
Applications Places System Wed Apr 17, 12:52
student@CN-R4: ~/tcp
File Edit View Search Terminal Help
this is the text file blah blah blah sfdlskjflskfjsklfsjl
2232
sfdlskjfslfs
sfdlskfjslfjskl
sfdlksjdfllksjflsjfjs
Hello
..
..
..
..
```

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]



Client Port : 57534, server use same port 5000 to send and listen here.

- 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]

Part2: server will bind the ip and port to the socket object to start Listening (waiting for incoming data). Then the server will accept the connection request from the client , starting to receive data and decoded data. Then send back digital information to the client. Client will first try to connect to the server by the given IP and port. Then encode data and send.

Part3: similar server build up process as part2, but it'll create a new file to store the contents that was sent in the client's file. Client will bind ip and port to the socket, then read file and send the byte stream. Once the server receives and stores the content, it will send back the confirmation code there.

- c) What does the `socket` system call return? [5 points]

A new socket object

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

Bind returns the status of the binding process, 0 for success, -1 for fail.

Server calls bind

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 does not have acknowledge packet(ACK) but TCP need ACK to guarantee the continuous stream, so the RTT of UDP is less than TCP which is faster under this scenario

f) What is Nagle's algorithm? What problem does it aim to solve and how? [10 points]

Nagle's Algorithm is to not send the new TCP segment when the window size or available sent data less than MSS, and the pipe still has unconfirmed data. Except conditions above all, it will send data. This Algorithm solve problem about an application repeatedly send data in small chunks through the method I describe above

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

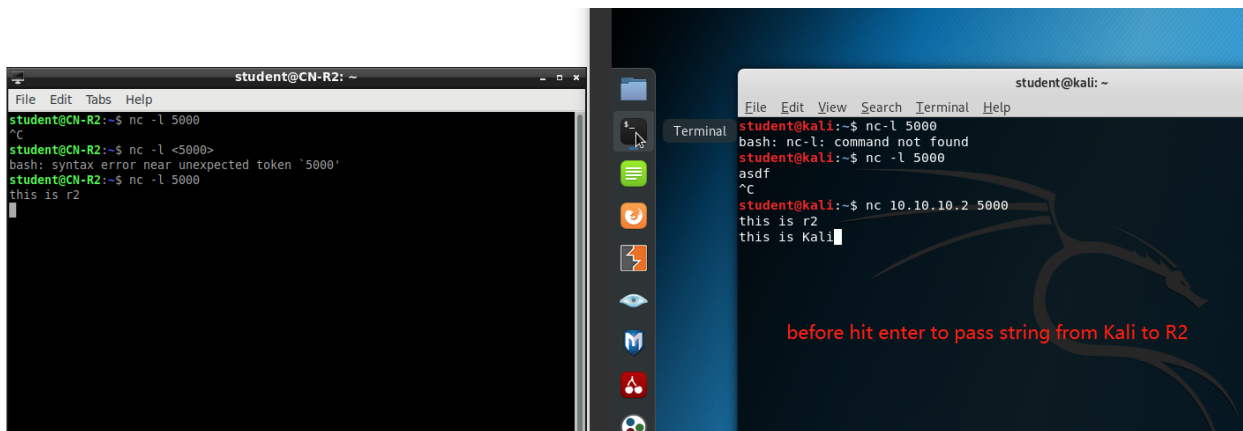
It will reduce the efficiency of data transmitting between sender and receiver.

Let's say two machines A and B, there is no data sent from B to A, and A is continuing to send data to B, and B uses delayed ACK here.

B will always have to wait for a certain amount of time to send back the ACK (delayed ACK), this action has significantly less efficiency than sending back ACK immediately

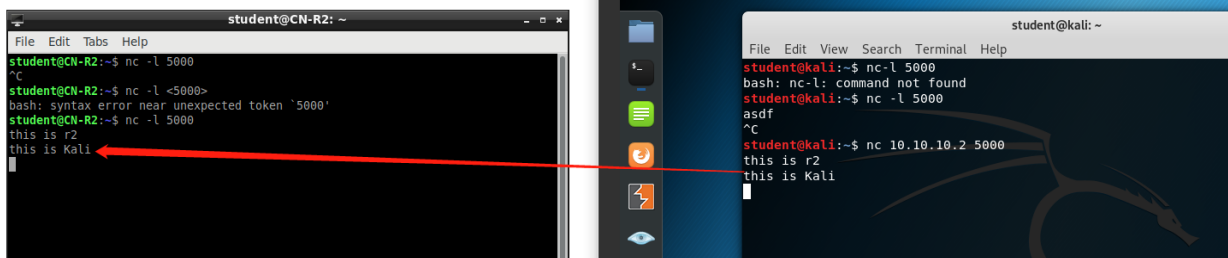
Submissions

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

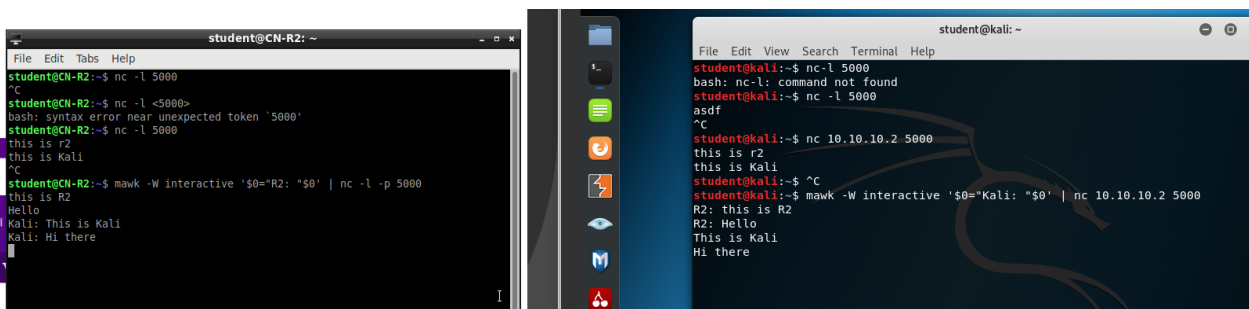


```
student@CN-R2: ~  
student@CN-R2:~$ nc -l 5000  
^C  
student@CN-R2:~$ nc -l <5000>  
bash: syntax error near unexpected token `5000'  
student@CN-R2:~$ nc -l 5000  
this is r2  
^C  
student@kali: ~  
student@kali:~$ nc -l 5000  
bash: nc-l: command not found  
student@kali:~$ nc -l 5000  
asdf  
^C  
student@kali:~$ nc 10.10.10.2 5000  
this is r2  
this is Kali
```

before hit enter to pass string from Kali to R2



```
student@CN-R2: ~  
student@CN-R2:~$ nc -l 5000  
^C  
student@CN-R2:~$ nc -l <5000>  
bash: syntax error near unexpected token `5000'  
student@CN-R2:~$ nc -l 5000  
this is r2  
this is Kali  
^C  
student@kali: ~  
student@kali:~$ nc -l 5000  
bash: nc-l: command not found  
student@kali:~$ nc -l 5000  
asdf  
^C  
student@kali:~$ nc 10.10.10.2 5000  
this is r2  
this is Kali
```



```
student@CN-R2: ~  
student@CN-R2:~$ nc -l 5000  
^C  
student@CN-R2:~$ nc -l <5000>  
bash: syntax error near unexpected token `5000'  
student@CN-R2:~$ nc -l 5000  
this is r2  
this is Kali  
^C  
student@CN-R2:~$ mawk -W interactive '$0="R2: "$0' | nc -l -p 5000  
Hello  
Kali: This is Kali  
Kali: Hi there  
^C  
student@kali: ~  
student@kali:~$ nc -l 5000  
bash: nc-l: command not found  
student@kali:~$ nc -l 5000  
asdf  
^C  
student@kali:~$ nc 10.10.10.2 5000  
this is r2  
this is Kali  
student@kali:~$ ^C  
student@kali:~$ mawk -W interactive '$0="Kali: "$0' | nc 10.10.10.2 5000  
R2: Hello  
This is Kali  
Hi there
```

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

```
import socketserver
print("-----Wait for Client Kali to Connect-----")
class MyTCP(socketserver.BaseRequestHandler):

    def data_p(self, data):
        count = 0
        digit = ''
        if 'SECRET' in data:
            for c in data:
                if c.isdigit():
                    count += 1
                    digit += c
            return 'Digits:' + digit + ', Counts: ' + str(count)
        else:
            return 'SECRET CODE NOT FOUND'

    def handle(self):
        self.data = self.request.recv(1024).strip()
        print('from Client: ', format(self.client_address[0]))
        print(self.data)
        self.request.sendall(bytes(self.data_p(self.data.decode()), 'UTF-8'))

if __name__ == '__main__':
    HOST, PORT = '10.10.10.2', 20001
    with socketserver.TCPServer((HOST, PORT), MyTCP) as server:
        server.serve_forever()
        server.server_close()
```

Applications ▾ Places ▾ Text Editor ▾ Mon 17:36

Open ▾ echo_tcp_client.py
~/Documents/hw6_socket

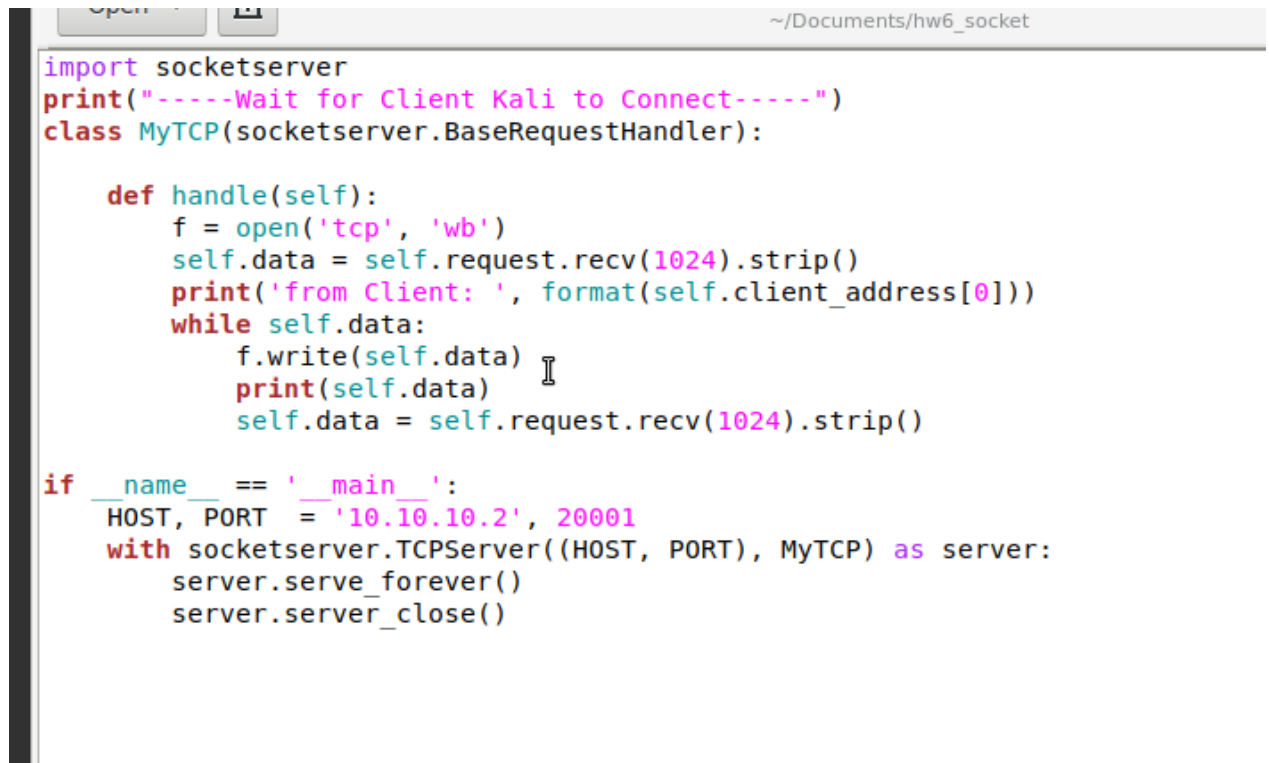
```
import socket
import sys

HOST, PORT = '10.10.10.2', 20001

data = "".join(sys.argv[1:])
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
    sock.connect((HOST, PORT))
    sock.sendall(bytes(data+ "\n" , "UTF-8"))
    rec = str(sock.recv(1024), "UTF-8")

print("Send: {}".format(data))
print("Received: {}".format(rec))
```


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



```
import socketserver
print("-----Wait for Client Kali to Connect-----")
class MyTCP(socketserver.BaseRequestHandler):

    def handle(self):
        f = open('tcp', 'wb')
        self.data = self.request.recv(1024).strip()
        print('from Client: ', format(self.client_address[0]))
        while self.data:
            f.write(self.data)
            print(self.data)
            self.data = self.request.recv(1024).strip()

if __name__ == '__main__':
    HOST, PORT = '10.10.10.2', 20001
    with socketserver.TCPServer((HOST, PORT), MyTCP) as server:
        server.serve_forever()
        server.server_close()
```



```
Applications ▾ Places ▾ Text Editor ▾ Mon 17:37
tcp_file_transfer_client.py
~/Documents/hw6_socket

import socket
import sys

HOST, PORT = '10.10.10.2', 20001

s = socket.socket()
s.connect((HOST, PORT))

with open('tcp.txt', 'rb') as f:
    l = f.read(1024)
    while l:
        s.send(l)
        print('Send:', repr(l))
        l = f.read(1024)
    f.close()
    print("Done Sending")
    s.close
```

4. Screenshots showing the behavior of Part 2. Make sure to include cases with and without the secret code. [5 points]

Set up server before transmit

```
student@CN-R2: ~/Documents/hw6_socket
File Edit Tabs Help
Connected By address ('10.10.10.3', 45632)
Wait for data ----
b'testwithSECRET\n'
testwithSECRET
Secret Code find here

Digits:testwithSECRET , Count: 14
^CTraceback (most recent call last):
  File "echo_tcp_server.py", line 10, in <module>
    conn, addr = s.accept() #build up connection
  File "/usr/lib/python3.6/socket.py", line 205, in accept
    fd, addr = self._accept()
KeyboardInterrupt
student@CN-R2:~/Documents/hw6_socket$ gedit echo_tcp_server.py

(gedit:1694): IBUS-WARNING **: 12:56:07.996: Unable to connect to ibus: Could not connect: Connection refused
student@CN-R2:~/Documents/hw6_socket$ gedit echo_tcp_server.py

(gedit:1729): IBUS-WARNING **: 13:03:22.615: Unable to connect to ibus: Could not connect: Connection refused
student@CN-R2:~/Documents/hw6_socket$ python3 echo_tcp_server.py
-----Wait for Client Kali to Connect-----
```

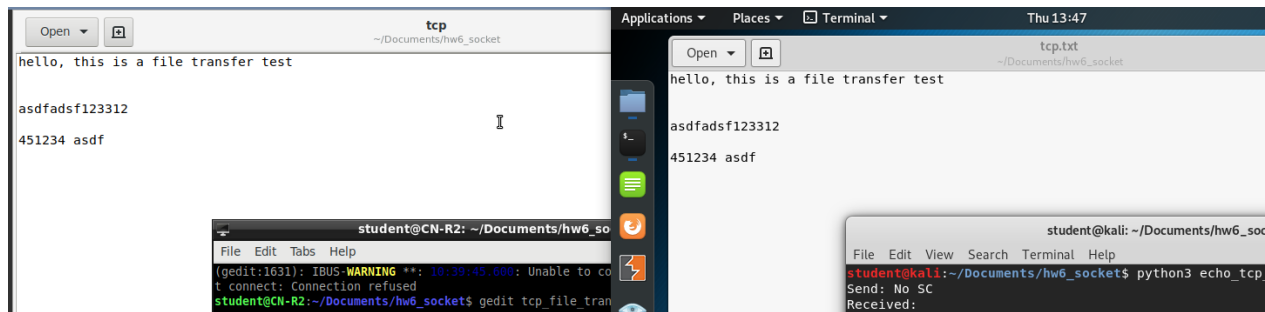
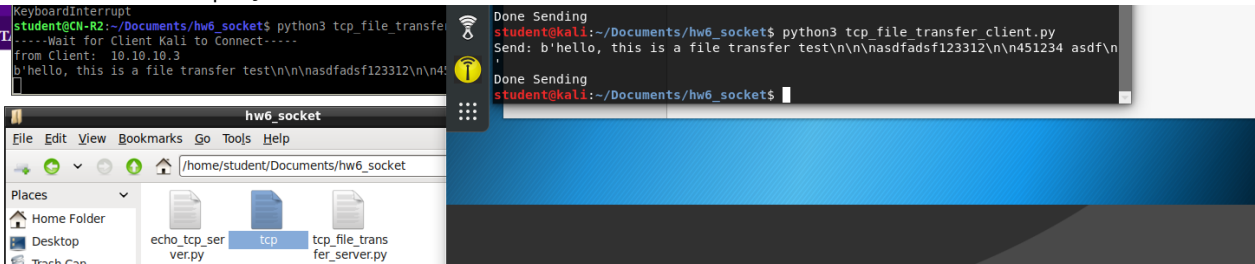
Different scenario test

```
student@CN-R2:~/Documents/hw6_socket$ python3 echo_tcp_server.py
-----Wait for Client Kali to Connect-----
from Client: 10.10.10.3
b'TEST with NO SC'
from Client: 10.10.10.3 [
b'TEST with SECRET CODE'
from Client: 10.10.10.3
b'TEST1 with2 SECRET3 CODE4'

student@kali:~/Documents/hw6_socket$ python3 echo_tcp_client.py SECRET 123 CODE 123
Send: SECRET 123 CODE 123
Received: Digits:123123, Counts: 6
student@kali:~/Documents/hw6_socket$ python3 echo_tcp_client.py TEST with NO SC
Send: TEST with NO SC
Received: SECRET CODE NOT FOUND
student@kali:~/Documents/hw6_socket$ python3 echo_tcp_client.py TEST with SECRET CODE
Send: TEST with SECRET CODE
Received: Digits:, Counts: 0
student@kali:~/Documents/hw6_socket$ python3 echo_tcp_client.py TEST1 with2 SECRET3 CODE4
Send: TEST1 with2 SECRET3 CODE4
Received: Digits:1234, Counts: 4
student@kali:~/Documents/hw6_socket$
```

5. 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]

Send file and display some information in the terminal



6. Answers to questions 4a-4g [60 points]

Please remember to submit your lab results as a single PDF document. While you may work in groups, you MUST submit your own work.