

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 R3's terminal. Type a message in R3'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.

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):

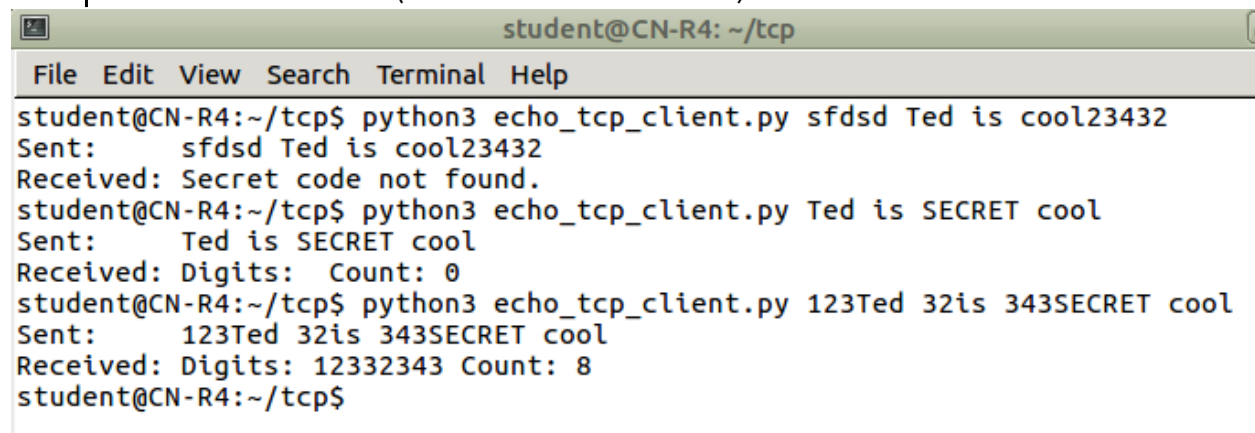
```
mawk -W interactive '$0="KALI: "$0' | nc <server IP> <port_number>
```

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)

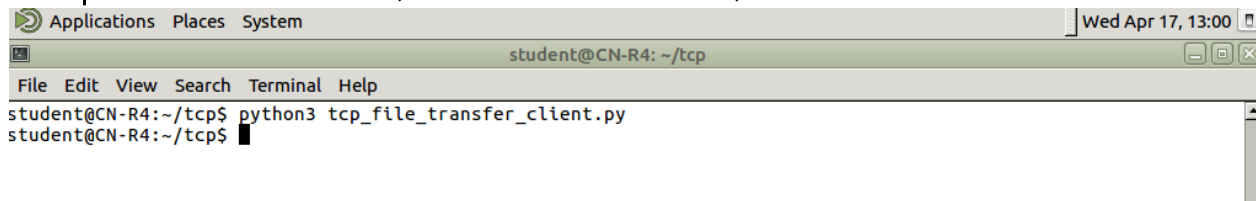


```
student@CN-R4: ~/tcp
File Edit View Search Terminal Help
student@CN-R4:~/tcp$ python3 echo_tcp_client.py sfdsd Ted is cool23432
Sent:      sfdsd Ted is cool23432
Received: Secret code not found.
student@CN-R4:~/tcp$ python3 echo_tcp_client.py Ted is SECRET cool
Sent:      Ted is SECRET cool
Received: 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$
```

Part 3: Client-server with file transfer

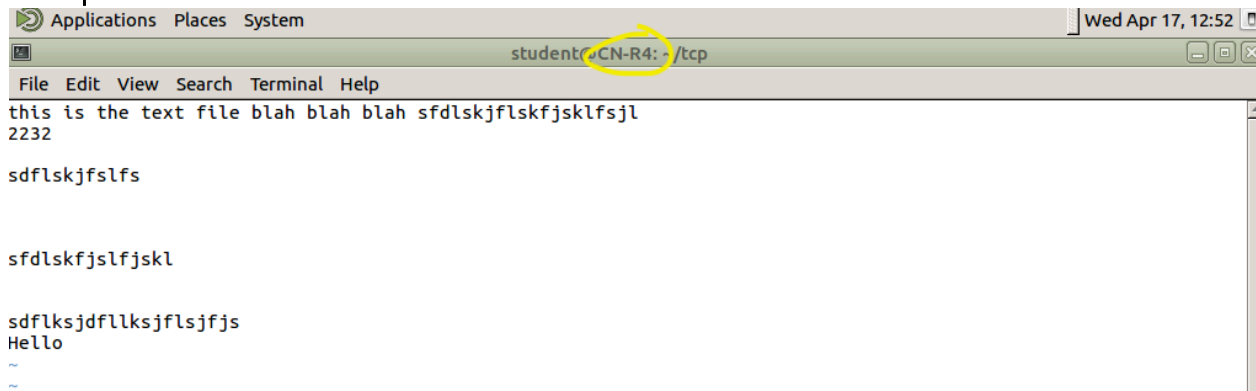
You should write two files for this part: **tcp_file_transfer_server.py** (on R2) and **tcp_file_transfer_client.py** (on KALI). The client should create a file (any text file with some content will suffice) and send the data in this file to the server. The server should receive the data and write it 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 console 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



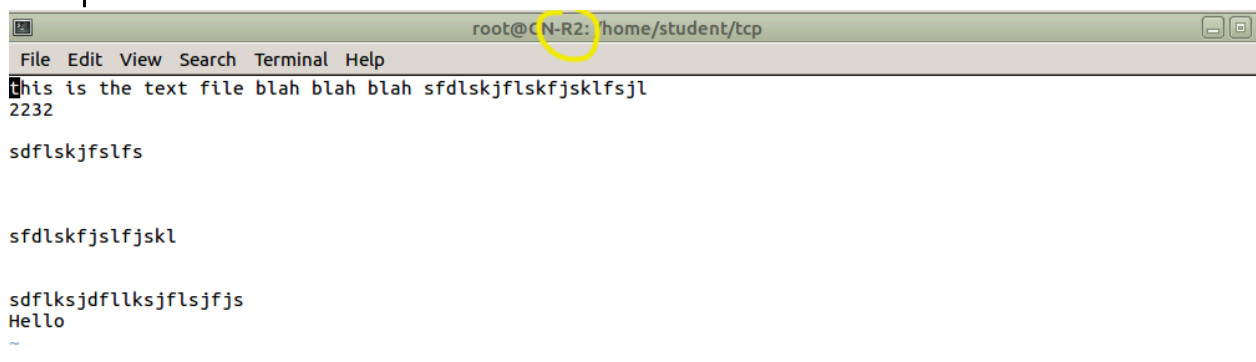
```
Applications Places System Wed Apr 17, 12:52
student@CN-R4: ~/tcp
File Edit View Search Terminal Help
this is the text file blah blah blah sfdlskjflskfjsklfsjl
2232

sfdlskjfslfs

sfdlskfjslfjskl

sfdlksjdfllksjflsjfjs
Hello
~
~
```

Example: file received



```
root@CN-R2: /home/student/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]
- 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]
- c) What does the `socket` system call return? [5 points]
- d) What does the `bind` system call return? Who calls `bind` (client/server)? [5 points]
- 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]
- f) What is Nagle's algorithm? What problem does it aim to solve and how? [10 points]
- g) Explain one potential scenario in which delayed ACK could be problematic. [10 points]

Submissions

- 1. Screenshots of R2 and KALI showing the `netcat` TCP chat [10 points]
- 2. Screenshots of `echo_tcp_server.py` and `echo_tcp_client.py` (showing all code) [10 points]
- 3. Screenshots of `tcp_file_transfer_server.py` and `tcp_file_transfer_client.py` (showing all code) [10 points]
- 4. Screenshots showing the behavior of Part 2. Make sure to include cases with and without the secret code. [5 points]
- 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]
- 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.