



Lab 0

Subjects	FSCT8561 - Security Applications
Date	@January 6, 2026

Course: FSCT 8561

Instructor: Dr. Maryam R. Aliabadi

Lab Duration: 3 hours

Overview

In Lab 0, students are introduced to the fundamentals of network programming using Python's socket library. The lab focuses on understanding basic network protocols and establishing simple client-server communication. Students will explore how sockets enable data exchange over TCP/IP networks, learning to create a basic server that can accept connections and a client that can send messages. Additionally, students will analyze the security implications of network communication.

Learning Objectives

- Understand the role of network protocols in computer communication
- Explain the client–server communication model
- Use Python sockets to establish basic TCP connections
- Analyze how protocol behavior affects security and reliability
- Demonstrate learning through explanation, screenshots, and code annotation

Required Reading

- **Mastering Python for Networking and Security** – Chapters 1 and 2

- <https://learning.oreilly.com/library/view/mastering-python-for/9781839217166/>
- Source code on: <https://github.com/PacktPublishing/Mastering-Python-for-Networking-and-Security-Second-Edition>
- **Python Official Documentation: socket module**
 - Official Python Documentation – socket Module
<https://docs.python.org/3/library/socket.html>
 - Real Python – Socket Programming in Python
<https://realpython.com/python-sockets/>
 - GeeksforGeeks – Socket Programming in Python
<https://www.geeksforgeeks.org/socket-programming-python/>
 - DigitalOcean – How To Use Sockets in Python 3
<https://www.digitalocean.com/community/tutorials/python-socket-programming>

Required Tools

- Python 3.9 or later
- Operating System: Windows, macOS, or Linux
- Code editor (VS Code recommended)
- Command-line access (Terminal or PowerShell)

Pre-Lab Preparation (Mandatory)

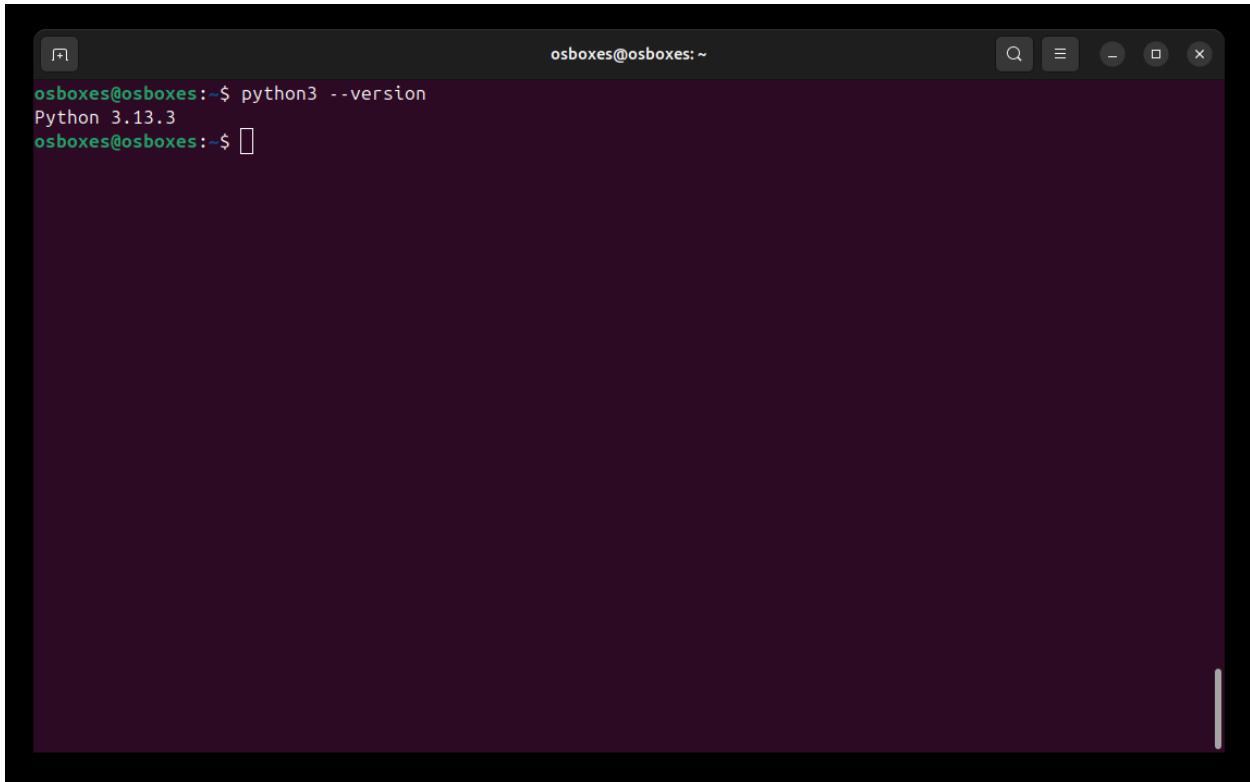
Before doing the lab, students must:

1. Read Chapters 1 and 2 of the textbook
2. Review the Python socket tutorial provided in this lab
3. Ensure Python is installed and accessible via command line

Lab Tasks

Task 1: Environment Verification

- Verify Python installation using `python --version`



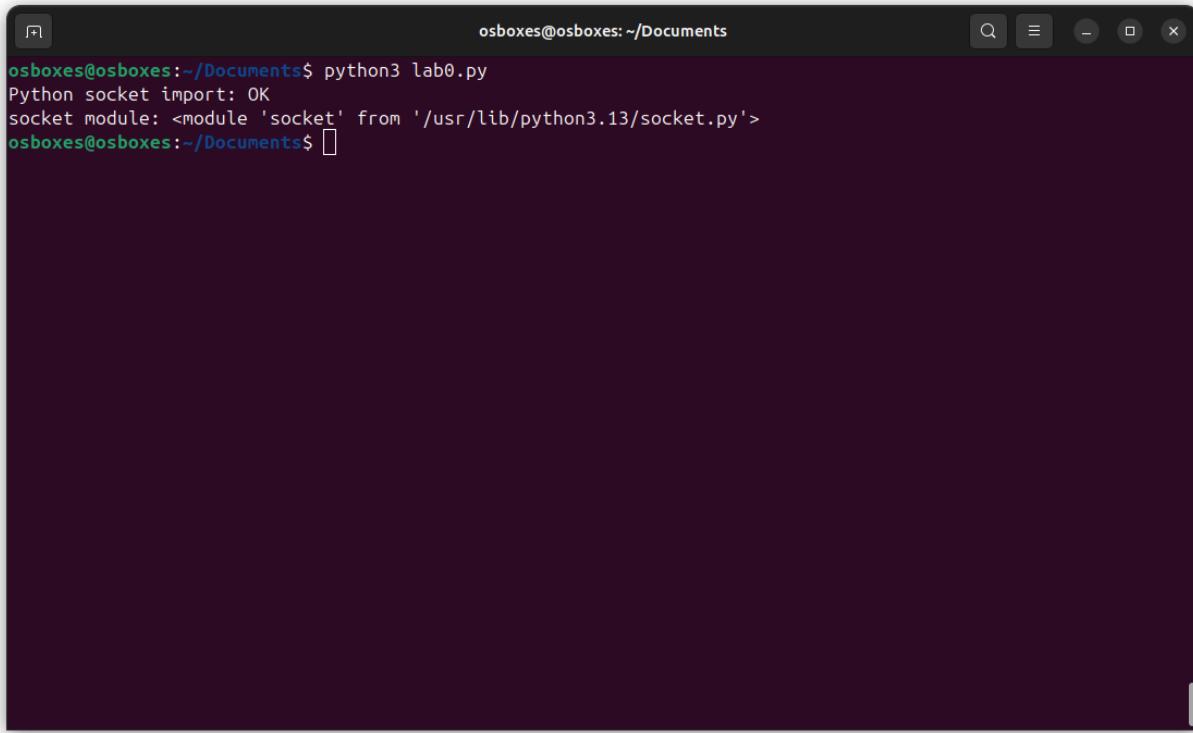
A screenshot of a terminal window titled "osboxes@osboxes: ~". The window shows the command "python3 --version" being run, which outputs "Python 3.13.3". The terminal has a dark background and a light-colored text area.

```
osboxes@osboxes:~$ python3 --version
Python 3.13.3
osboxes@osboxes:~$
```

- Import the socket module without errors

```
import socket

print("Python socket import: OK")
print("socket module:", socket) # show where socket module is imported from
```



A terminal window titled "osboxes@osboxes: ~/Documents". The command "python3 lab0.py" is run, and the output shows the socket module imported successfully from "/usr/lib/python3.13/socket.py".

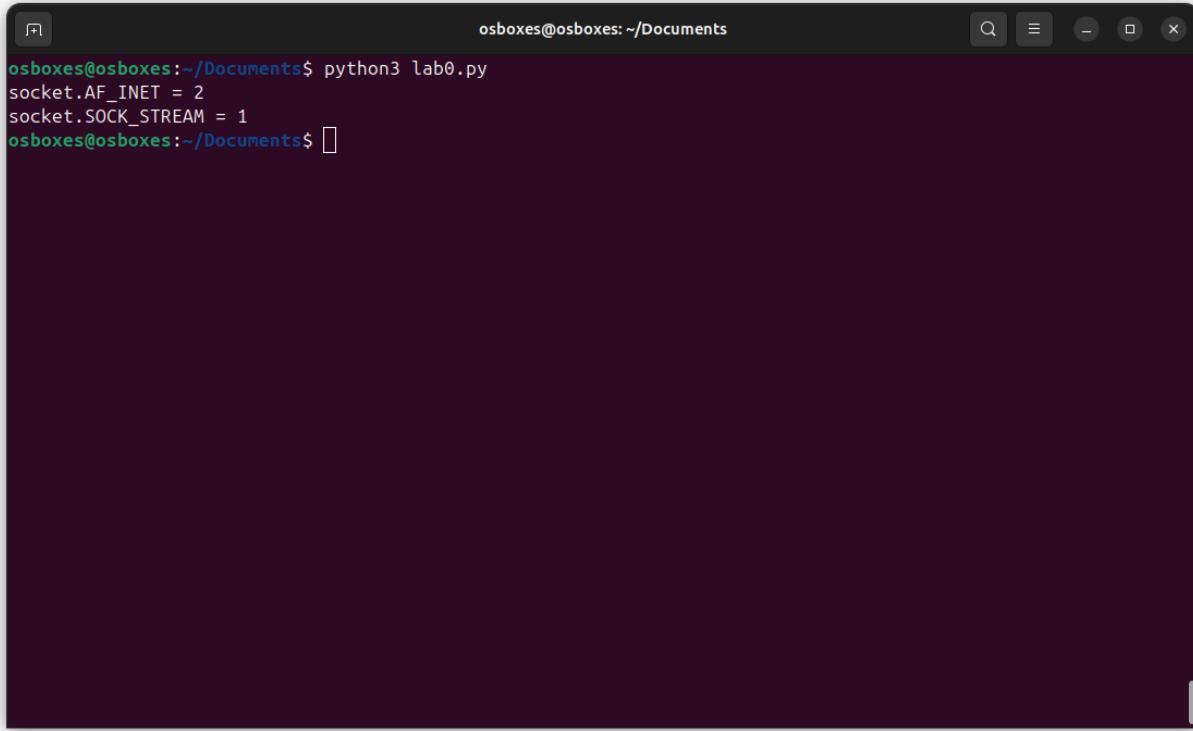
```
osboxes@osboxes:~/Documents$ python3 lab0.py
Python socket import: OK
socket module: <module 'socket' from '/usr/lib/python3.13/socket.py'>
osboxes@osboxes:~/Documents$ 
```

Task 2: Understanding Sockets

- Create a simple Python script that imports `socket`
- Identify and explain the purpose of `socket.AF_INET` and `socket.SOCK_STREAM`

```
import socket

print("socket.AF_INET =", socket.AF_INET) # print value of IPv4 (2)
print("socket.SOCK_STREAM =", socket.SOCK_STREAM) # print the value for
TCP
```



A terminal window titled "osboxes@osboxes: ~/Documents". The command "python3 lab0.py" is run, displaying the output:

```
osboxes@osboxes:~/Documents$ python3 lab0.py
socket.AF_INET = 2
socket.SOCK_STREAM = 1
osboxes@osboxes:~/Documents$
```

Task 3: Simple Client Connection

- Write a Python script that creates a TCP socket
- Connect to a known public server (example: `example.com` on port `80`)
- Send a simple HTTP GET request and print the response
- Capture and explain the connection behavior

```
import socket

HOST = "example.com"
PORT = 80

# build http get req.
request = (
    "GET / HTTP/1.1\r\n"
    f"Host: {HOST}\r\n"
    "Connection: close\r\n"
    "\r\n"
```

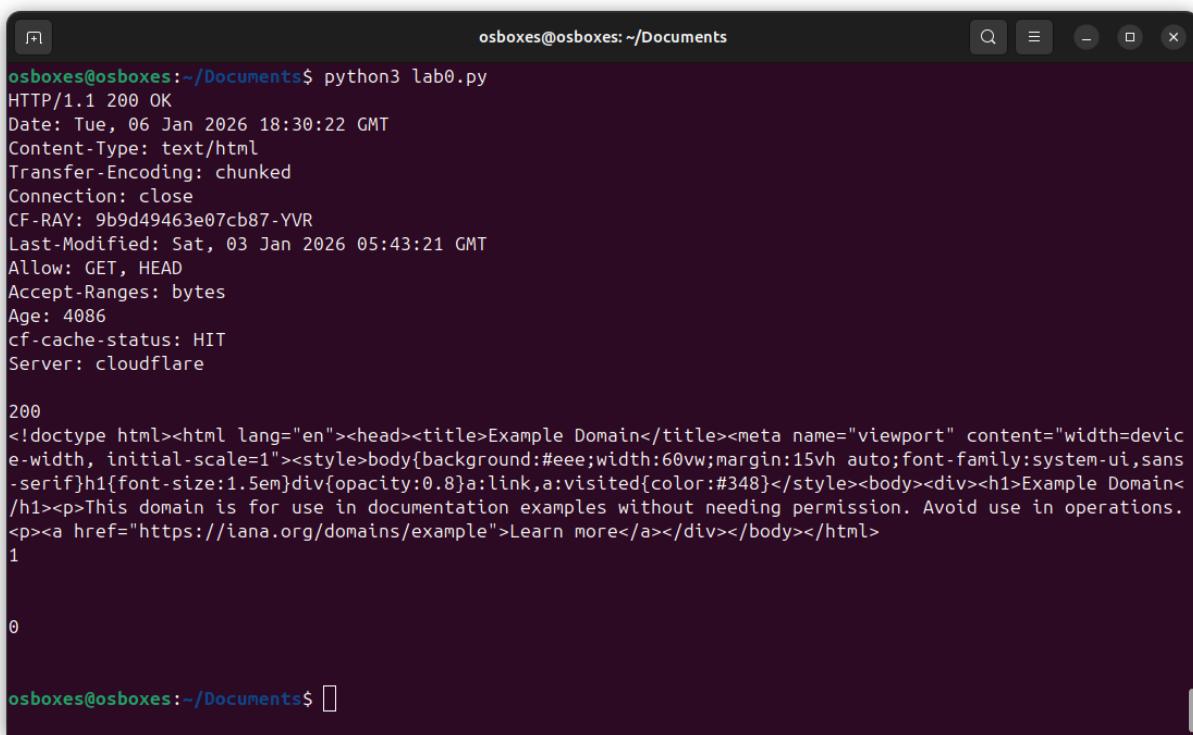
```

)
# define ipv4 + tcp socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT)) # connect to remote
    s.sendall(request.encode("utf-8"))

    data = b"" # init empty buffer
    while True:
        chunk = s.recv(4096) # max 4096 bytes
        if not chunk:
            break
        data += chunk

    print(data.decode("utf-8", errors="replace")) # print data

```



A terminal window titled "osboxes@osboxes: ~/Documents" is shown. The command "python3 lab0.py" is run, and the output is displayed. The output includes HTTP headers such as "HTTP/1.1 200 OK", "Date: Tue, 06 Jan 2026 18:30:22 GMT", and "Content-Type: text/html". It also shows a "200" status code and an HTML response body containing a title, a note about the domain being for documentation examples, and a link to the IANA example domain page.

```

osboxes@osboxes:~/Documents$ python3 lab0.py
HTTP/1.1 200 OK
Date: Tue, 06 Jan 2026 18:30:22 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: close
CF-RAY: 9b9d49463e07cb87-YVR
Last-Modified: Sat, 03 Jan 2026 05:43:21 GMT
Allow: GET, HEAD
Accept-Ranges: bytes
Age: 4086
cf-cache-status: HIT
Server: cloudflare

200
<!doctype html><html lang="en"><head><title>Example Domain</title><meta name="viewport" content="width=device-width, initial-scale=1"><style>body{background:#eee; width:60vw; margin:15vh auto; font-family:system-ui,sans-serif}h1{font-size:1.5em}div{opacity:0.8}a:link,a:visited{color:#348}</style></head><body><div><h1>Example Domain</h1><p>This domain is for use in documentation examples without needing permission. Avoid use in operations.<p><a href="https://iana.org/domains/example">Learn more</a></p></div></body></html>
1

0

osboxes@osboxes:~/Documents$ 

```

Task 4: Create a TCP Server

Write a Python script that:

1. Binds to `localhost` on a chosen port (e.g., `12345`)
2. Listens for incoming connections
3. Accepts a connection and receives a message from the client
4. Sends a response back

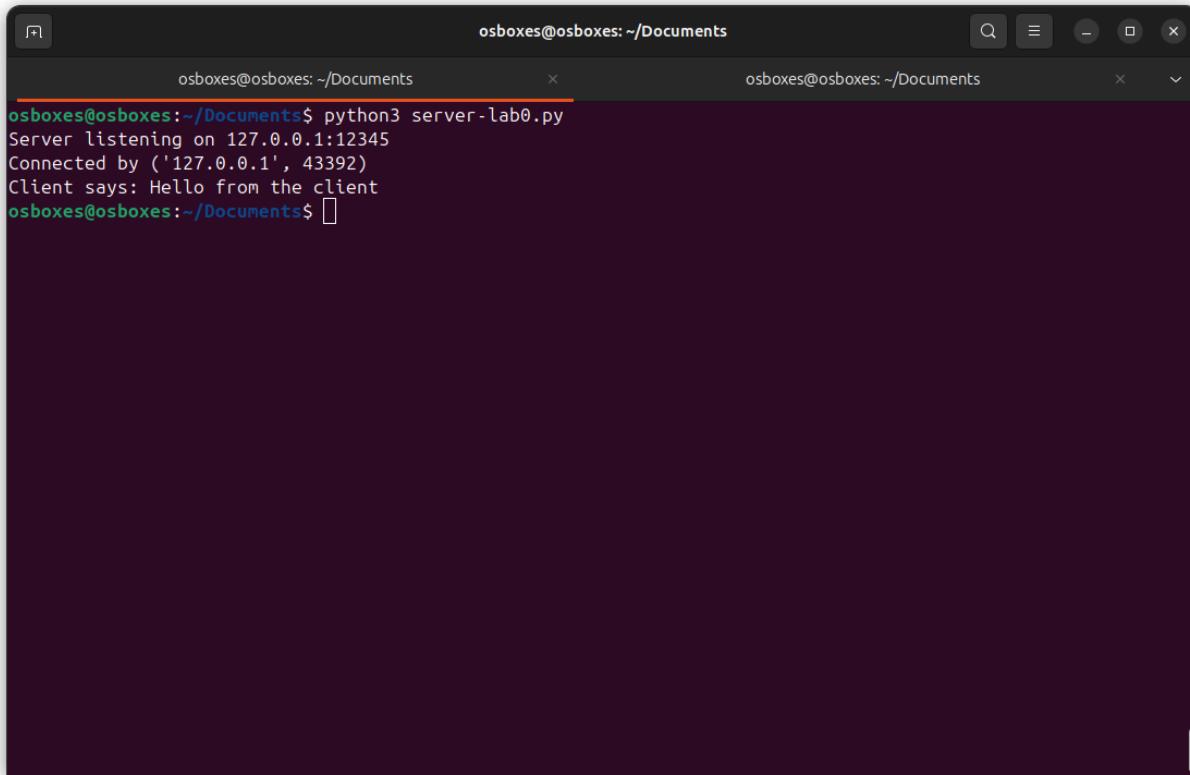
```
import socket

HOST = "127.0.0.1"
PORT = 12345

# initialize IPv4 + TCP socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server:
    server.bind((HOST, PORT))
    server.listen(1) # allow up to 1 client
    print(f"Server listening on {HOST}:{PORT}")

    conn, addr = server.accept() # block until client connects, get a new socket
    with conn: # ensure socket is closed properly
        print("Connected by", addr) # view clients
        msg = conn.recv(1024).decode("utf-8", errors="replace") # receive 1024
        bytes
        print("Client says:", msg)

        reply = "Message received by server"
        conn.sendall(reply.encode("utf-8")) # encode reply and send back to cli
        nt
```



```
osboxes@osboxes: ~/Documents
osboxes@osboxes: ~/Documents
osboxes@osboxes: ~/Documents$ python3 server-lab0.py
Server listening on 127.0.0.1:12345
Connected by ('127.0.0.1', 43392)
Client says: Hello from the client
osboxes@osboxes: ~/Documents$
```

Task 5: Create a TCP Client

Write a Python script that:

1. Connects to your server on `localhost:12345`
2. Sends a custom message
3. Receives and prints the server's response

```
import socket

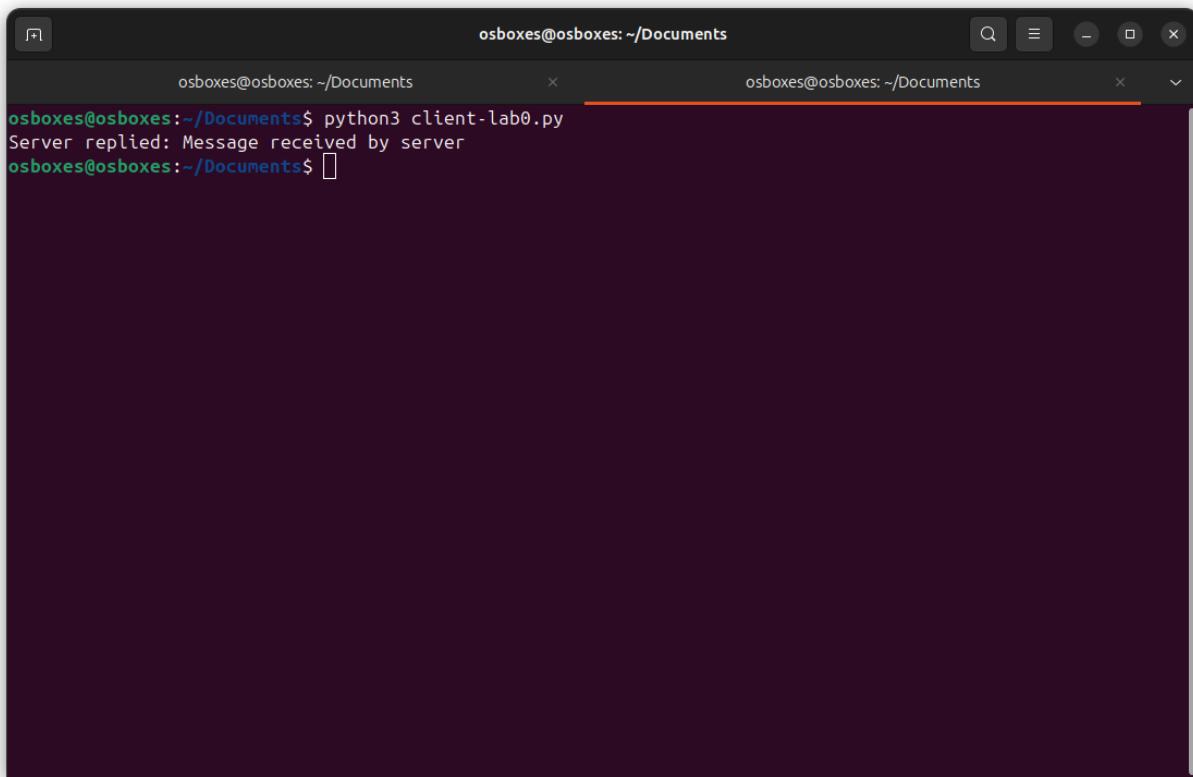
HOST = "127.0.0.1"
PORT = 12345

message = "Hello from the client" # message

# create a client socket with IPv4 + TCP
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client:
    client.connect((HOST, PORT)) # connect to server
```

```
client.sendall(message.encode("utf-8")) # encode the message

reply = client.recv(1024).decode("utf-8", errors="replace") # receive up to 1
024 bytes from the server
print("Server replied:", reply) # servers reply
```



The screenshot shows a terminal window with two tabs. The left tab is titled 'osboxes@osboxes: ~/Documents' and contains the command 'osboxes@osboxes:~/Documents\$ python3 client-lab0.py'. The right tab is also titled 'osboxes@osboxes: ~/Documents' and displays the output: 'Server replied: Message received by server'. The terminal has a dark background and light-colored text.

Task 5: Security Reflection

- Identify at least three security risks related to raw socket programming.
 - Plaintext data can be sniffed or altered
 - No authentication, open listening ports allow unauthorized connections
 - Resource exhaustion, attackers can also DoS the service by sending oversized or malformed data
- Explain why input validation, access control, and protocol awareness matter at the network level.

- Network input is untrusted and TCP is just a byte stream, so we need to validate size and format and handle message framing correctly. Access control limits who can connect so random hosts cannot misuse the service.
- Suggest ways to secure your simple client-server application
 - Use plaintext only on localhost: bind the server to 127.0.0.1 so only your computer can connect.
 - Limit what you accept: if the message is too long or empty, close the connection.
 - Add a shared password: client sends a fixed secret first, server only responds if it matches.

Deliverables

Students must submit a single PDF file containing:

1. Answers to reflection questions
2. Annotated Python code snippets
3. Screenshots showing successful execution
4. A short paragraph explaining what was learned and what was unclear
5. submit your report with the following name format through the Learning Hub.

Filename: Lab0-Ashwin-CharathSandran-A01240798.pdf

Good luck!