**1. Problem Statement**

**Given the SMB server IP 98.81.69.119 and the shared path "shared", mount it using cli commands and find the task file related the socket programming**

 Ans :

sudo apt install cifs-tools

mount -t cifs //**98.81.69.119**/shared /mnt/smb

**2. Problem Statement**

**Task #1: The Missing Message Mystery**

**Scenario**

**Alice tries to send "HELLO WORLD" to Bob’s server. However, Bob only receives part of the message.**

**As the detective, your job is to investigate why the message is incomplete and fix the issue so Bob receives the full string.**

Ans : Fixing buffer\_size in client\_socket.recv(8) to client\_socket.recv(12)

**Code :**

**import** socket

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind((**'localhost'**,9999))

server.listen(1)

print(**"Bob's server is listening"**)

client\_socket , address = server.accept()

message = client\_socket.recv(12)

print(f**"Bob received : '{message.decode()}'"**)

client\_socket.close()

**Task #2: The Port Scanner Problem**

**Scenario**

**A detective always checks locked and unlocked doors. Similarly, you will create a simple port scanner that tests whether a given port on a host is open or closed.**

**💡 Hints:**

**Use socket.connect().**

**Wrap the connection attempt inside try/except blocks.**

**Code :**

**import** socket

detector = socket.socket()

host = input(**"Host : "**)

port = int(input(**"Port : "**))

detector.settimeout(2)

**try**:

detector.connect((host,port))

print(f**"Port {port} on {host} is Open"**)

**except**:

print(f**"Port {port} on {host} is Closed"**)

**Output:**

* python3 port-scanner.py

Host : localhost

Port : 23

Port 23 on localhost is Closed

**Task #3: The Curl Detective Mystery**

**Scenario**

**Detectives often record every clue exactly as found. Similarly, you will build a Python server( server.py) that accepts HTTP requests sent via curl and prints the raw request exactly as received.**

**Important notes:**

**Use only socket programming**

**You do not need to send a response back to curl.**

**Code :**

**import** socket

server = socket.socket()

server.bind((**'localhost'**,9090))

server.listen(1)

client, addr = server.accept()

request = client.recv(1024);

print(request.decode())

client.close()

**Output :**

> curl -X POST http://localhost:9090/test -d "Detective at work"

curl: (52) Empty reply from server

> python3 server3.py

POST /test HTTP/1.1

Host: localhost:9090

User-Agent: curl/8.5.0

Accept: \*/\*

Content-Length: 17

Content-Type: application/x-www-form-urlencoded

Detective at work

**3. Problem Statement**

**You are provided with a bastion host that has a public IP address and a .pem key for SSH, along with a private application server that can only be accessed through the bastion. A MySQL RDS instance is also running inside the private subnet. You are expected to first connect to the bastion using the provided .pem file and then attempt to securely SSH into the private application server from there. Once inside the private instance, verify connectivity to the given RDS endpoint the MySQL client.**

**Note: As part of the task, you are expected to troubleshoot and resolve any issues you encounter while establishing SSH connectivity to the private instance or while reaching the MySQL RDS instance from it.**

**Credentials :**

Public IP - 35.175.122.215

Private IP - 10.0.2.201

username – ubuntu

**RDS Database**:

Hostname - *networking-lab-mysql.cuukwis7t1js.us-east-1.rds.amazonaws.com*

User *- admin*

Password *- LabPassword123!*

Port *– 3306*

**In bastion host :**

Delete these ufw rules : sudo ufw delete [rule number]

22/tcp DENY OUT Anywhere

22/tcp (v6) DENY OUT Anywhere (v6)

ssh -i intern-key.pem [ubuntu@10.0.2.201](mailto:ubuntu@10.0.2.201)

**In Private Server :**

Delete these ufw rules : sudo ufw delete [rule number]

3306/tcp DENY OUT Anywhere

3306/tcp (v6) DENY OUT Anywhere (v6)

mysql -h networking-lab-mysql.cuukwis7t1js.us-east-1.rds.amazonaws.com -u admin -p

**4. Problem Statement**

**Develop two simple applications running on different ports, for example one on port 5001 and another on port 5002, each returning a distinct response. Configure an Application Load Balancer (ALB) on port 8080 with path-based routing so that requests to /api are directed to the first application and requests to /app are directed to the second. Since the ALB may become a bottleneck, add a Network Load Balancer (NLB) using HAProxy in front of it, operating at Layer 4. The expected flow is:**

**Flow: Client → NLB (L4) → ALB (L7) → Web servers**

**Configure the NLB in TCP mode and test the behavior by sending requests through it. Use tools such as curl -v and tcpdump to observe and capture the traffic flow. Compare the behavior with direct access to the ALB.**

1. **Analyze why inconsistencies appear when the NLB is in TCP mode and HTTP mode**
2. **Finally, update the configuration to HTTP Mode and explain your findings**

Ans : sudo apt install haproxy

/etc/haproxy/haproxy.cfg :

frontend alb\_frontend

bind \*:8080

mode http

acl is\_api path\_beg /api

acl is\_app path\_beg /app

use\_backend app1\_backend if is\_api

use\_backend app2\_backend if is\_app

backend app1\_backend

mode http

http-request set-path %[path,regsub(^/api,/)]

server app1 127.0.0.1:5001 check

backend app2\_backend

mode http

http-request set-path %[path,regsub(^/app,/)]

server app2 127.0.0.1:5002 check

frontend nlb\_frontend

bind \*:8000

mode tcp

default\_backend alb\_backend

backend alb\_backend

server alb 127.0.0.1:8080

mode http

In NLB TCP mode, It forwards raw TCP streams. There are chances for forwarding partial or badly timed segments of headers and body.

In NTB HTTP mode, It can parse headers and make any required changes to headers. HTTP supports path based routing.

Resources :

https://www.linode.com/docs/guides/linux-mount-smb-share/