Haibin Lai Computer Network Assignment 1

TSNA- A Telnet like Service with simplified NTLM Authentication

Author: Haibin Lai,
Student ID: 12211612,

Structure

Introduction

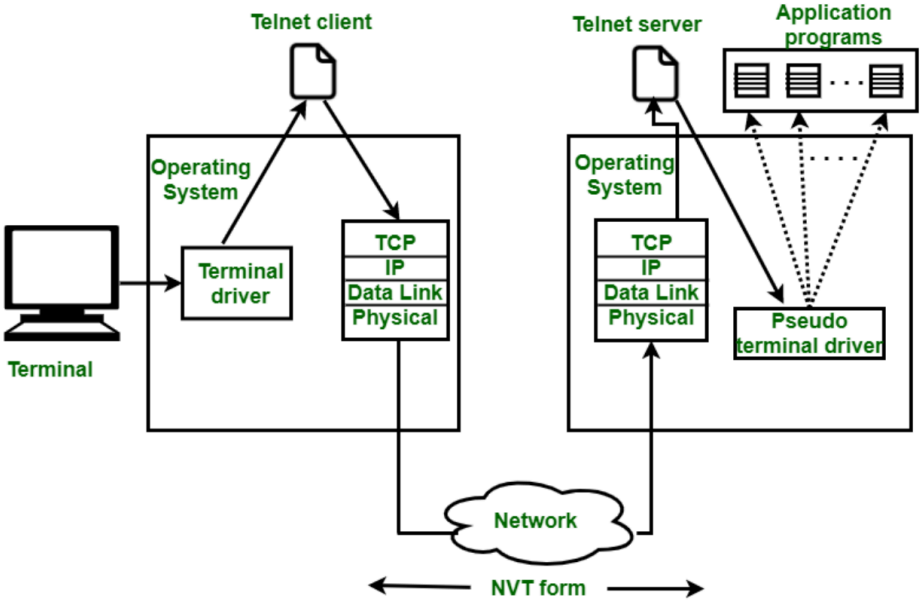
Command Output

- Task1
- Task2
- Task3
- Task4
- Task5

Exception

Introduction on Telnet

Telnet is a network protocol primarily used to establish text-based, bidirectional communication between a local computer and a remote server. It operates over a TCP/IP connection, allowing users to remotely control servers via command-line interface. Telnet is often used to manage devices, servers, or network equipment, but due to its lack of encryption, it poses security risks and has largely been replaced by encrypted protocols like SSH.



A figure showing how Telnet works on Client/Server

Command Output

The following figure shows the output of each command in system.

Task1: Connection Establishment

The client can try to connect server.
Note:
Not Allowed IP :

1. not in 32bits.(`inet_aton` convert an IP address in string format (123.45.67.89) to the 32-bit packed binary format used in low-level network functions.
2. 0.0.0.0: it represents "all network interfaces" or "local address" but cannot be used to connect to a remote host.
   Special Allowed IP:
3. localhost

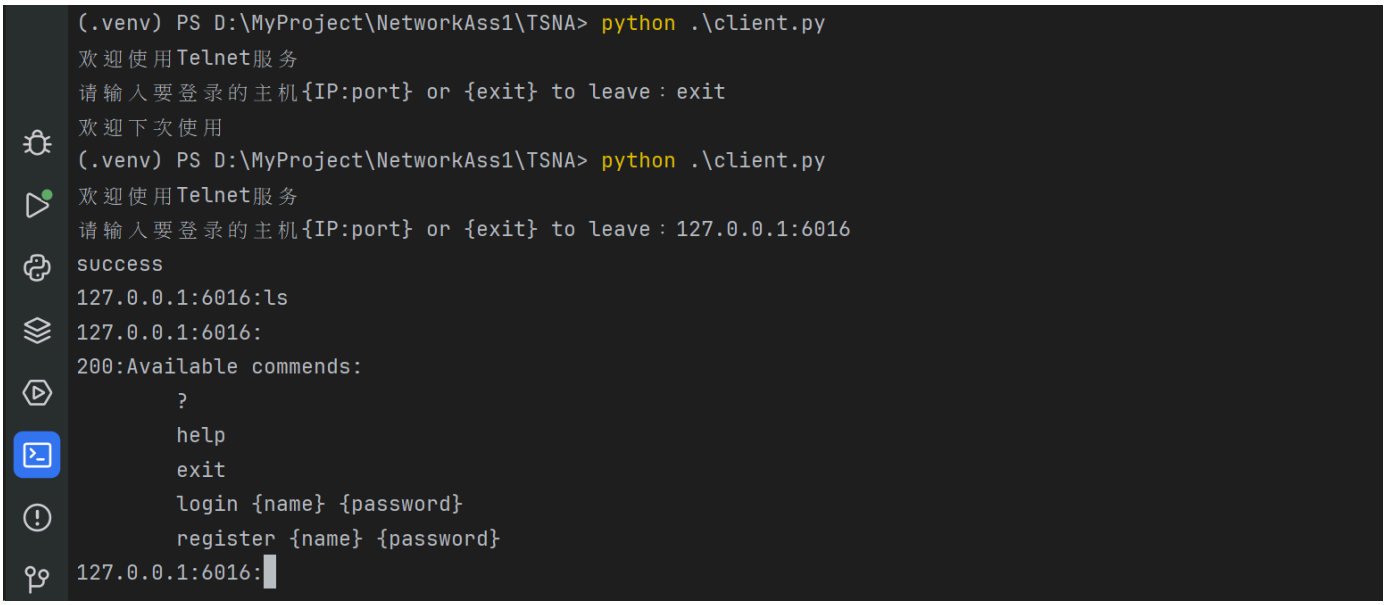
When we first start our sesrver, it starts a Thread TCP Server. It can handle multiple clients.

```
if __name__ == '__main__':
    server_address = host
    server_port = port
    # Create the server
    with socketserver.ThreadingTCPServer((host, port), TSNAServerHandler) as server:
        print("serving at host", server_address, "with port: ", server_port)
        # Activate the server;
        try:
            server.serve_forever()
        except KeyboardInterrupt:
            print("Shutting down by keyboard")
```

The server will wait for TCP connection. It will listen for the specfic `host` and `port`. When the data is got, it will enter `main_loop` with `client_address` and `login_user`.

```
class TSNAServerHandler(socketserver.BaseRequestHandler):
    def handle(self):
        print('Got connection from', self.client_address)
        sc_socket = self.request
        if sc_socket is not None:
            sc_socket.send("success".encode('UTF-8'))
        else:
            print("Fail to catch a socket")
        login_user = ''
        is_continue = True

        # get data
        while is_continue:
            is_continue, login_user = (
                main_loop(socket_conn=sc_socket, client_address=self.client_address,
login_user=login_user))
```

Now the client can be able to send a ip:port to establish a TCP connection.



Next, on the `main_loop`, it will handle recv data and save it into `data_log.txt`.

```
22  def main_loop(socket_conn: socket, client_address, login_user):
23      """
24
25      :param socket_conn: socket connection
26      :param client_address: client IP address
27      :param login_user: str current logged-in user
28      :return continue flag: boolean for main loop continue judgement, login user: str
29      """
30      ## Task 1.3
31      # TODO: finish the codes
32
33      receive_data = ''
34      try:
35          receive_data = socket_conn.recv(1024).decode('utf-8')
36      except ConnectionAbortedError:
37          print("Connection aborted from "+client_address)
38          return False, None
39
40      print("Received data:", receive_data, " from ", client_address)
41      data_log = str(client_address)
42      with open("data_log.txt", 'a') as file:
43          file.write(data_log+","+str(receive_data)+","+str(time.time())+"\n")
44      file.close()
45      ## Task 1.3
46
47      # Command processing before login
48      if not login_user:
49          # Command processing without arguments
50          if receive_data == '?' or receive_data == 'help' or receive_data == 'ls':
51              feedback_data = 'Available commends: \n\t' + '\n\t'.join(commands)
52              feedback_data = SUCCESS(feedback_data)
53          elif receive_data == 'exit':
54              feedback_data = 'disconnected'
```

```
223 ('127.0.0.1', 41852),login haibin 910f4aff69c6c642b3756489bb81a945,1731163555.6838896
224 ('127.0.0.1', 49117),ls,1731165889.9536746
225
```

Task 2: User Authentication

At the beginning of initializing server, it will load user-password from `user_records_txt`.

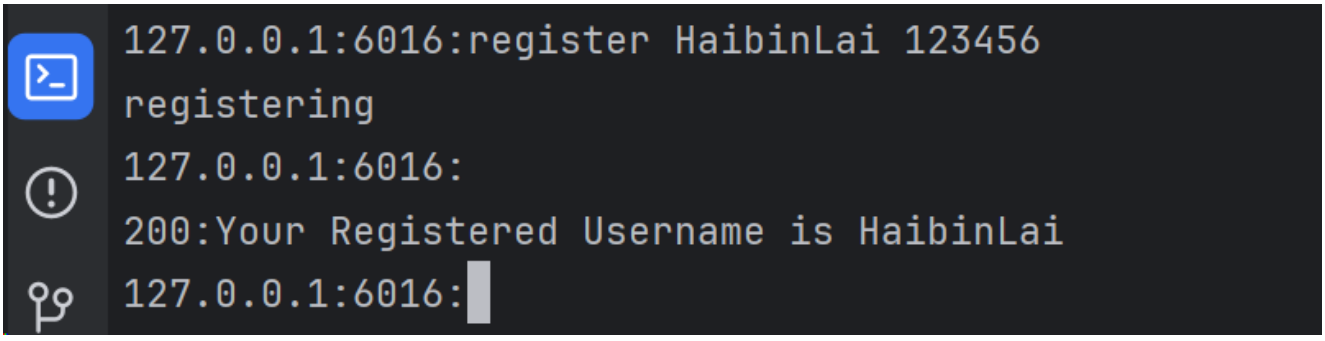
```
def load_users(user_records_txt):
    """
    Task 2.1 Load saved user information (username and password)    :param user_records_txt: a txt file containing username and password records    :return users: dict {'username':'password'}
"""    # Done: finish the codes
    users = {}  # Initialize

    if not os.path.exists(user_records_txt):
        file = open(user_records_txt, 'w')
```

```
        file.close()

    with open(user_records_txt, 'r') as user_r:
        for line in user_r:
            line = line.strip()
            if line:  # Ensure the line is not empty
                username, password = line.split(':', 1)  # Split
                users[username] = password  # Store

    return users
```

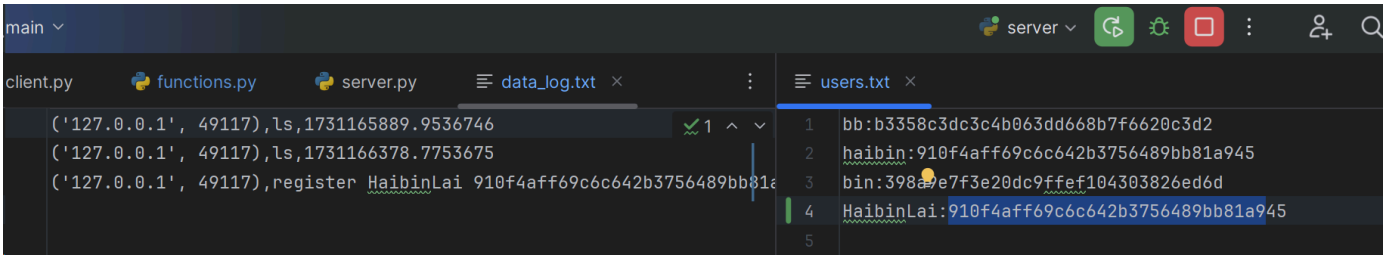
And client can register a new user with new password.

```
def user_register(cmd, users):
    """
    Task 2.2 Register command processing    :param cmd: Instruction string    :param users: The
dict to hold information about all users    :return feedback message: str    """    # done: finish
the codes
    new_username = cmd[1]
    new_password = cmd[2]

    # Attempt to register a user that is already registered
    if new_username in users:
        print("Username is already in users!")
        return FAILURE("Username is already in users!")
    users[new_username] = new_password
    with open(user_inf_txt, 'a') as user_r:
        user_r.write(new_username+":"+new_password+"\n")

    return SUCCESS("Your Registered Username is " + new_username)
```

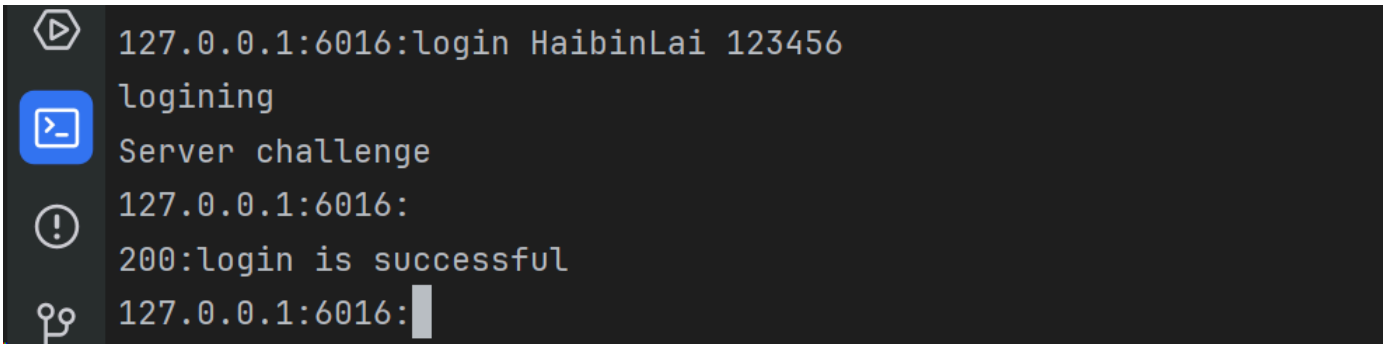
And it will be registered in server.



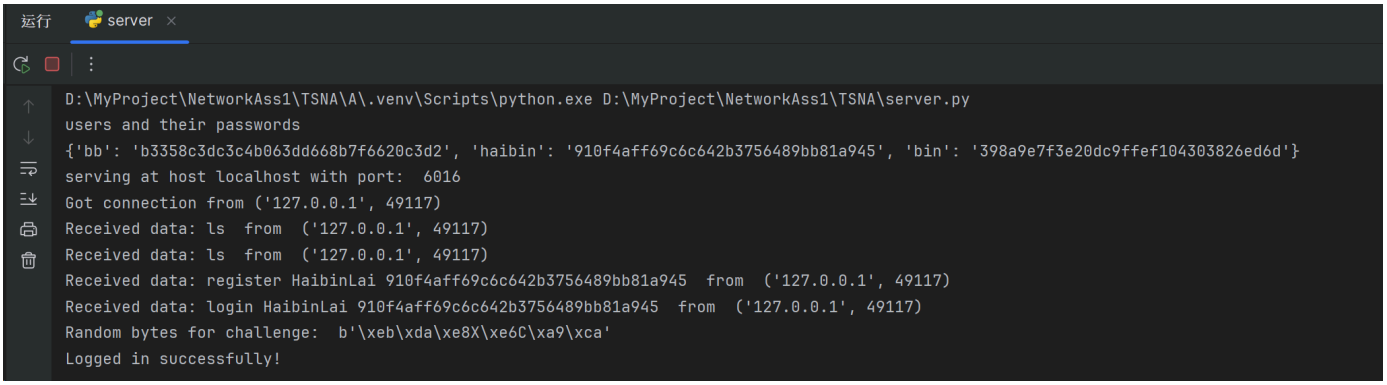
Both the data log and user file will not record the password in **plaintext** . Instead, they will be recorded as **Ciphertext**.



Now the user can login to its record.

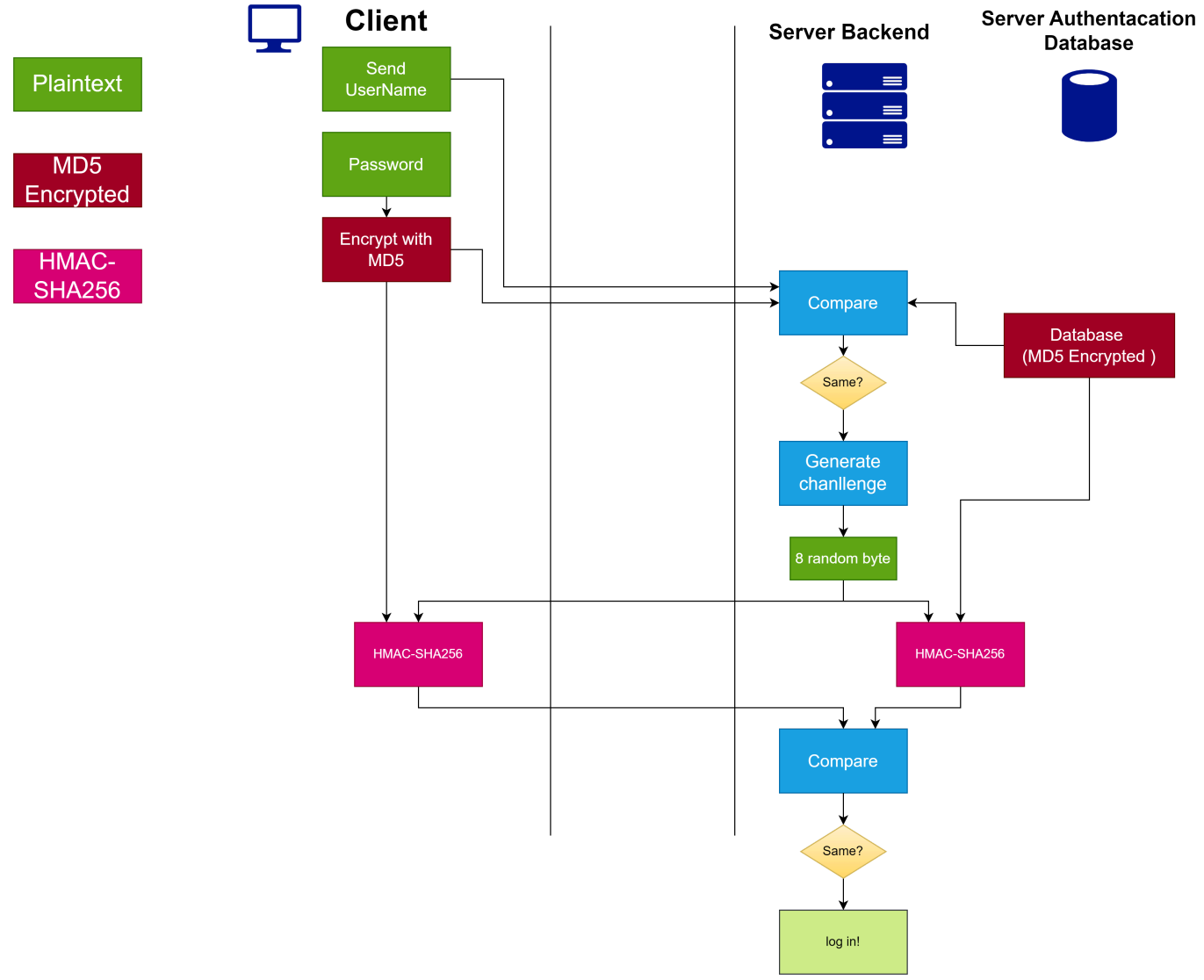


The server side shows the logging procedure. We will explain this logs on NTLM Authentication.



Task 3: NTLM Authentication

The procedure of NTLM authenrication as the assignment request is shown as following figure.



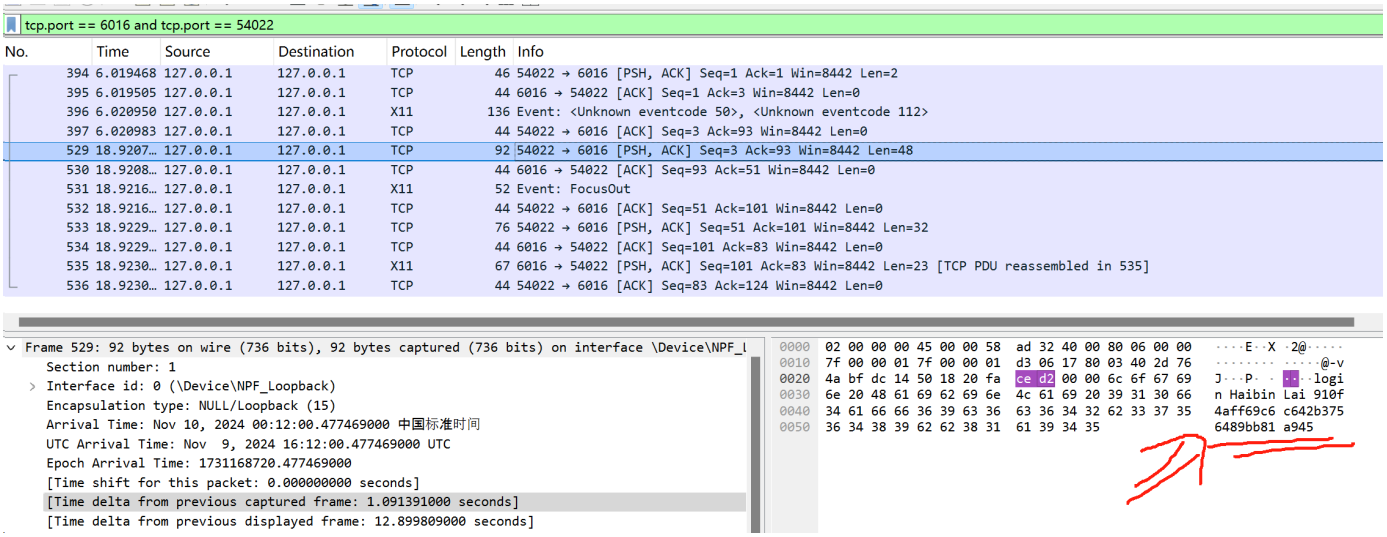

```
D:\MyProject\NetworkAss1\TSNA\A\.venv\Scripts\python.exe D:\MyProject\NetworkAss1\TSNA\server.py
users and their passwords
{'bb': 'b3358c3dc3c4b063dd668b7f6620c3d2', 'haibin': '910f4aff69c6c642b3756489bb81a945', 'bin': '398a9e7f3e20
serving at host localhost with port:  6016
Got connection from ('127.0.0.1', 54022)
Received data: ls  from  ('127.0.0.1', 54022)
Received data: login HaibinLai 910f4aff69c6c642b3756489bb81a945  from  ('127.0.0.1', 54022)
Random bytes for challenge:  b'\x9c\x0c$\xa6\xca\xafLY'
Logged in successfully!
```

So, when first send the message, password will be encrypted in function `ntlm_hash_func` .

```
def ntlm_hash_func(password):
    """
    This function is used to encrypt passwords by the MD5 algorithm    """    # 1. Convert
password to hexadecimal format
    hex_password = ''.join(format(ord(char), '02x') for char in password)

    # 2. Unicode encoding of hexadecimal passwords
    unicode_password = hex_password.encode('utf-16le')
```

```
    # 3. The MD5 digest algorithm is used to Hash the Unicode encoded data
    md5_hasher = hashlib.md5()
    md5_hasher.update(unicode_password)

    # Returns the MD5 Hash
    return md5_hasher.hexdigest()
```

As the wireshark shows, it contains `user_name` HaibinLai in plain context and encrypted password.



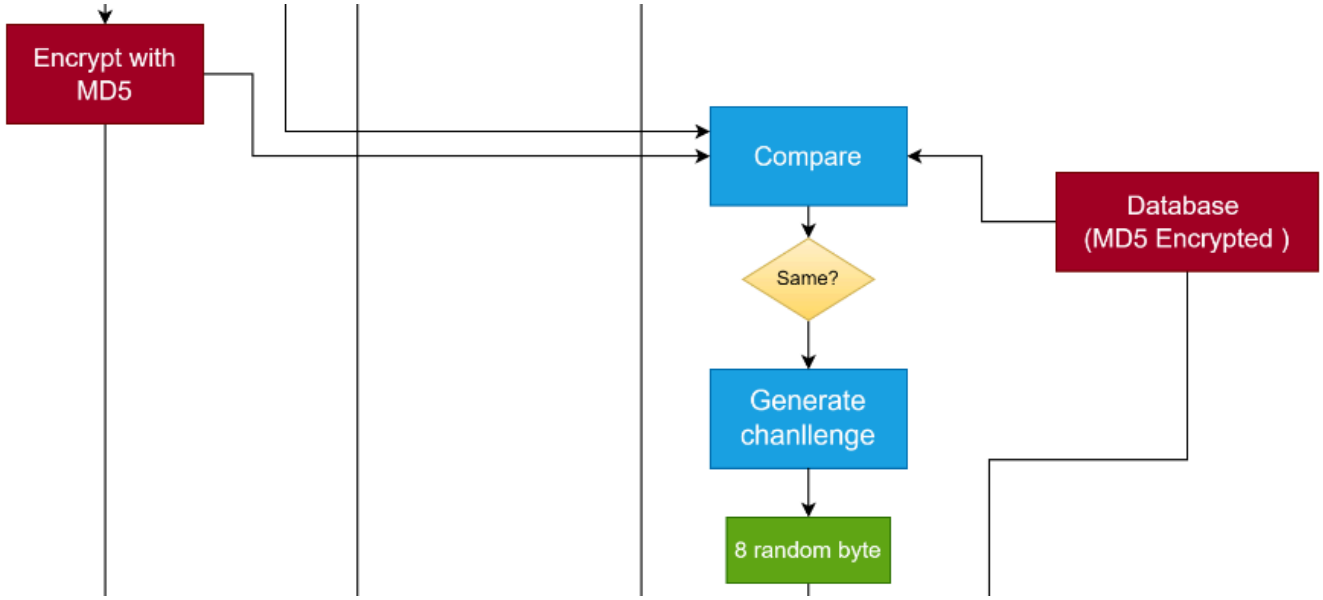
tcp.port == 6016 and tcp.port == 54022

No.	Time	Source	Destination	Protocol	Length	Info
394	6.019468	127.0.0.1	127.0.0.1	TCP	46	54022 → 6016 [PSH, ACK] Seq=1 Ack=1 Win=8442 Len=2
395	6.019505	127.0.0.1	127.0.0.1	TCP	44	6016 → 54022 [ACK] Seq=1 Ack=3 Win=8442 Len=0
396	6.020950	127.0.0.1	127.0.0.1	X11	136	Event: <Unknown eventcode 50>, <Unknown eventcode 112>
397	6.020983	127.0.0.1	127.0.0.1	TCP	44	54022 → 6016 [ACK] Seq=3 Ack=93 Win=8442 Len=0
529	18.9207…	127.0.0.1	127.0.0.1	TCP	92	54022 → 6016 [PSH, ACK] Seq=3 Ack=93 Win=8442 Len=48
530	18.9208…	127.0.0.1	127.0.0.1	TCP	44	6016 → 54022 [ACK] Seq=93 Ack=51 Win=8442 Len=0
531	18.9216…	127.0.0.1	127.0.0.1	X11	52	Event: FocusOut
532	18.9216…	127.0.0.1	127.0.0.1	TCP	44	54022 → 6016 [ACK] Seq=51 Ack=101 Win=8442 Len=0
533	18.9229…	127.0.0.1	127.0.0.1	TCP	76	54022 → 6016 [PSH, ACK] Seq=51 Ack=101 Win=8442 Len=32
534	18.9229…	127.0.0.1	127.0.0.1	TCP	44	6016 → 54022 [ACK] Seq=101 Ack=83 Win=8442 Len=0
535	18.9230…	127.0.0.1	127.0.0.1	X11	67	6016 → 54022 [PSH, ACK] Seq=101 Ack=83 Win=8442 Len=23 [TCP PDU reassembled in 535]
536	18.9230…	127.0.0.1	127.0.0.1	TCP	44	54022 → 6016 [ACK] Seq=83 Ack=124 Win=8442 Len=0

```
Frame 529: 92 bytes on wire (736 bits), 92 bytes captured (736 bits) on interface \Device\NPF_L
    Section number: 1
  > Interface id: 0 (\Device\NPF_Loopback)
    Encapsulation type: NULL/Loopback (15)
    Arrival Time: Nov 10, 2024 00:12:00.477469000 中国标准时间
    UTC Arrival Time: Nov  9, 2024 16:12:00.477469000 UTC
    Epoch Arrival Time: 1731168720.477469000
    [Time shift for this packet: 0.000000000 seconds]
    [Time delta from previous captured frame: 1.091391000 seconds]
    [Time delta from previous displayed frame: 12.899809000 seconds]
```

```
0000  02 00 00 00 45 00 00 58  ad 32 40 00 80 06 00 00   ····E··X ·2@·····
0010  7f 00 00 01 7f 00 00 01  d3 06 17 80 03 40 2d 76   ·········· ·····@-v
0020  4a bf dc 14 50 18 20 fa  ce d2 00 00 6c 6f 67 69   J···P· · ····logi
0030  6e 20 48 61 69 62 69 6e  4c 61 69 20 39 31 30 66   n Haibin Lai 910f
0040  34 61 66 66 36 39 63 36  63 36 34 32 62 33 37 35   4aff69c6 c642b375
0050  36 34 38 39 62 62 38 31  61 39 34 35              6489bb81 a945
```

Then, the server will determine if the password in MD5 are the same in Database MD5:



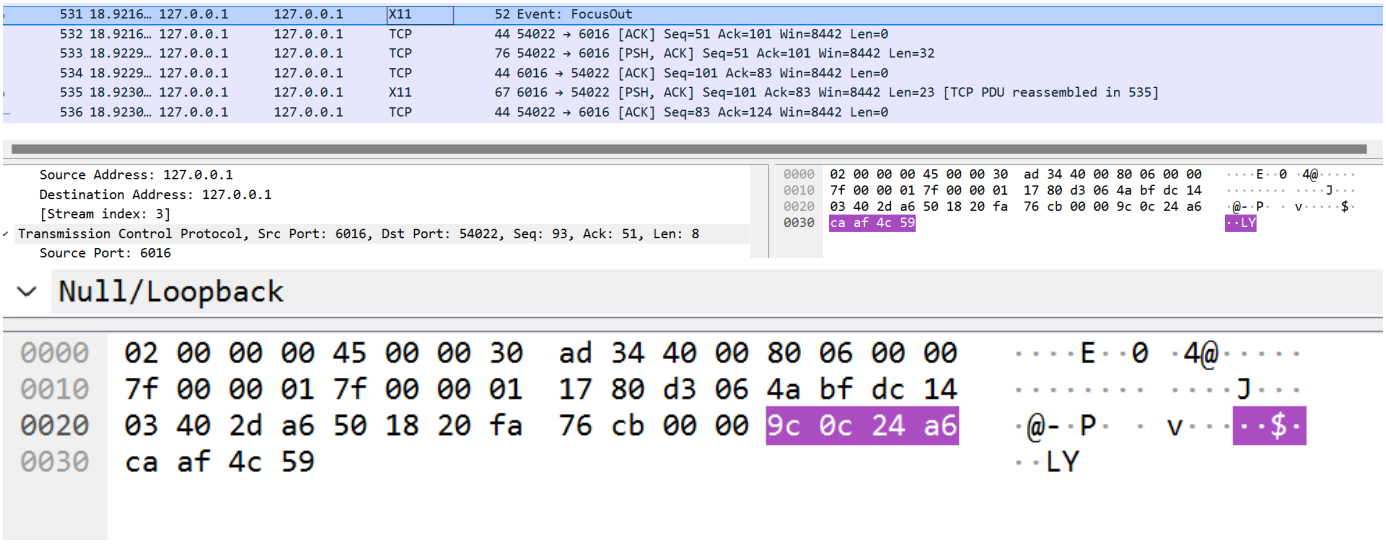

```
def login_authentication(conn, cmd, users):
    login_user = cmd[1]
    login_password = cmd[2]
    if login_user in users:
        if users[login_user] == login_password:
            challenge = generate_challenge()  # a challenge
            conn.send(challenge)

            calcu = conn.recv(1024)
            ans = calculate_response(login_password, challenge)
            if ans == calcu:
```

```
                print("Logged in successfully!")
                return SUCCESS("login is successful"), login_user
            else:
                print("Authentication Failed!")
                return FAILURE("Authentication Failed!"), None
        else:
            return FAILURE("Wrong password!"), None
    else:
        return FAILURE("The user does not exist!"), None
```

Next, both client and server will check the `HMAC-SHA256` encrypted code with encrypted password with 8 random byte sent by Server.

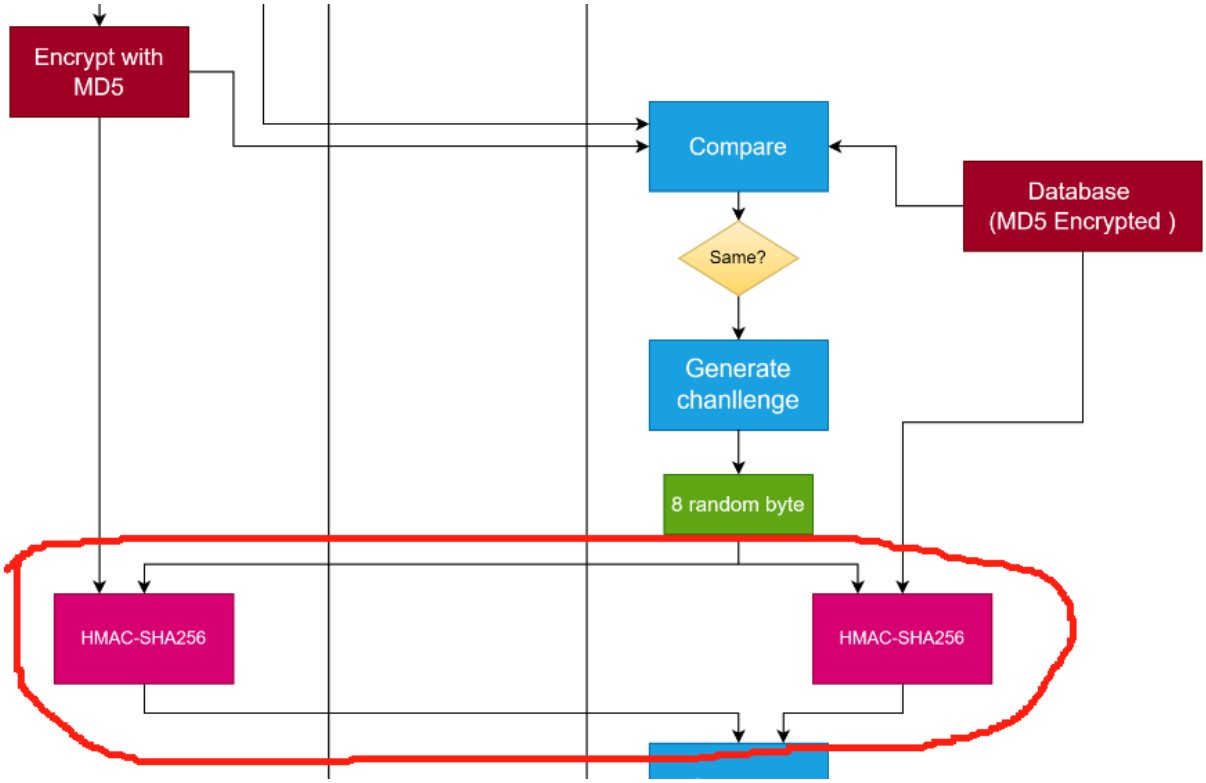
```
def generate_challenge():
    """
    Task 3.2    :return information: bytes random bytes as challenge message    """    # done:
finish the codes
    random_bytes = os.urandom(8)
    print("Random bytes for challenge: ", random_bytes)
    return random_bytes
```

Here we find the 8 byte: `9c 0c 24 a6 ca af 4c 59` . (Here `\xafLY` means `4c 59` are `LY` in `ASCII`) . As for X11 in Wireshark, it's because it's it default detecting protocol.




```
Got connection from ('127.0.0.1', 54022)
Received data: ls  from  ('127.0.0.1', 54022)
Received data: login HaibinLai 910f4aff69c6c642b3756489bb81a945  from  ('127.0.0.1', 54022)
Random bytes for challenge:  b'\x9c\x0c$\xa6\xca\xafLY'
Logged in successfully!
```

After getting 8 byte challenge, both client and server encrypted with `HMAC-SHA256`.



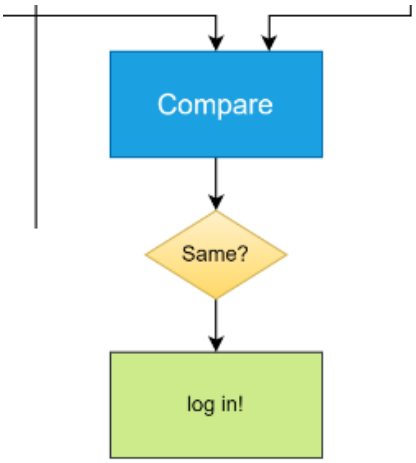

```python
def calculate_response(ntlm_hash: str, challenge: bytes):
    # 假设 ntlm_hash 是一个字符串，使用 utf-8 编码转换为字节
    ntlm_hash_bytes = ntlm_hash.encode('utf-8') if isinstance(ntlm_hash, str) else
bytes(ntlm_hash)

    # 假设 challenge 是字节串，直接使用
    # 如果 challenge 也是字符串，需要转换为字节
    challenge_bytes = challenge if isinstance(challenge, bytes) else challenge.encode('utf-8')
    return hmac.new(ntlm_hash_bytes, msg=challenge_bytes, digestmod=hashlib.sha256).digest()
```

Then the encrypted code in client side will send to Server to check whether it pass the challenge.

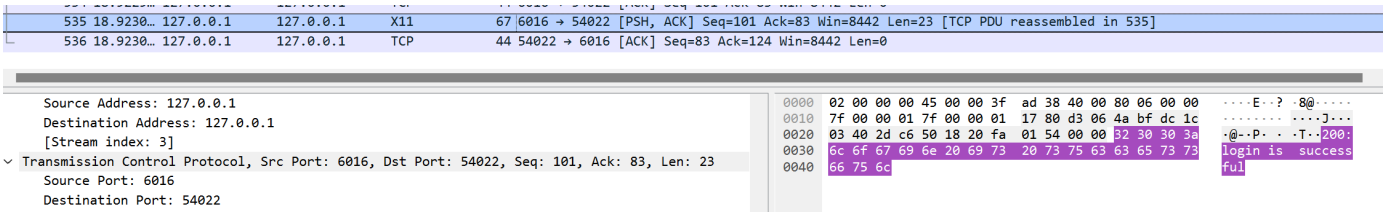
```
533 18.9229… 127.0.0.1  127.0.0.1  TCP   76 54022 → 6016 [PSH, ACK] Seq=51 Ack=101 Win=8442 Len=32
534 18.9229… 127.0.0.1  127.0.0.1  TCP   44 6016 → 54022 [ACK] Seq=101 Ack=83 Win=8442 Len=0
535 18.9230… 127.0.0.1  127.0.0.1  X11   67 6016 → 54022 [PSH, ACK] Seq=101 Ack=83 Win=8442 Len=23 [TCP PDU reassembled in 535]
536 18.9230… 127.0.0.1  127.0.0.1  TCP   44 54022 → 6016 [ACK] Seq=83 Ack=124 Win=8442 Len=0
```

```
    Source Address: 127.0.0.1
    Destination Address: 127.0.0.1
    [Stream index: 3]
Transmission Control Protocol, Src Port: 54022, Dst Port: 6016, Seq: 51, Ack: 101, Len: 32
    Source Port: 54022
```

```
0000  02 00 00 00 45 00 00 48  ad 36 40 00 80 06 00 00   ····E··H ·6@·····
0010  7f 00 00 01 7f 00 00 01  d3 06 17 80 03 40 2d a6   ········ ·····@-·
0020  4a bf dc 1c 50 18 20 fa  d2 36 00 00 7a b4 af b2   J···P· · ·6··z···
0030  b4 0a eb f9 13 3f 29 a1  10 5c 7a 58 12 49 ee 16   ·····?)· ·\zX·I··
0040  16 a0 51 f8 4a a7 d7 b2  6c c3 f2 19               ··Q·J··· l···
```



If success, it will login.



Task 4: Command Processing

`sum $(num1) $(num2) ...`

note that it's limited to `float` format.

```
127.0.0.1:6016:sum 1 2 3 4 5
127.0.0.1:6016:
200:15.0
127.0.0.1:6016:sum 1.4 2.5
127.0.0.1:6016:
200:3.9
127.0.0.1:6016:sum -45 15.34
127.0.0.1:6016:
200:-29.66
127.0.0.1:6016:sum 0.2 0.1
127.0.0.1:6016:
200:0.30000000000000004
127.0.0.1:6016:
```

`multiply $(num1) $(num2) ...`

```
127.0.0.1:6016:multiply 2 3
127.0.0.1:6016:
200:6.0
127.0.0.1:6016:multiply 2 0
127.0.0.1:6016:
200:0.0
127.0.0.1:6016:multiply -1 -1
127.0.0.1:6016:
200:1.0
127.0.0.1:6016:multiply 223.4 5.3
127.0.0.1:6016:
200:1184.02
127.0.0.1:6016:
```

`sub $(num1) $(num2)`

`subtract $(num1) $(num2)`

```
127.0.0.1:6016:sub 3 5
127.0.0.1:6016:
200:-2.0
127.0.0.1:6016:sub 123 24
127.0.0.1:6016:
200:99.0
127.0.0.1:6016:sub 123 23 45
127.0.0.1:6016:
400:Please enter Valid number! subtract $(number1) $(number2)
127.0.0.1:6016:subtract 23 45
127.0.0.1:6016:
200:-22.0
```

login logout help, changepwd

```
127.0.0.1:6016:login HaibinLai 123456
logining
Server challenge
127.0.0.1:6016:
200:login is successful
127.0.0.1:6016:help
127.0.0.1:6016:
200:Available commands:
        ?
        help
        exit
        logout
        changepwd {newpassword}
        sum [a] [b] ...
        sub [a] [b]
        multiply [a] [b] ...
        divide [a] [b]
127.0.0.1:6016:changepwd 123
changepwding
127.0.0.1:6016:
200:Successfully changed password
127.0.0.1:6016:logout
127.0.0.1:6016:
200:Logout from current user: HaibinLai
127.0.0.1:6016:login HaibinLai 123
logining
Server challenge
127.0.0.1:6016:
200:login is successful
127.0.0.1:6016:
```

Exception

1. TCP Connection

IP Address

Not Allowed IP :

1. not in 32bits.(`inet_aton` convert an IP address in string format (123.45.67.89) to the 32-bit packed binary format used in low-level network functions.
   `Invalid IP Port: ", server_port`
2. 0.0.0.0: it represents "all network interfaces" or "local address" but cannot be used to connect to a remote host.
   `Invalid IP for 0.0.0.0, it represent all network interfaces.`
3. Invalid ip without correct form of "ip:port"
   `Invalid IP and port format. Please use ip:port with ipv4 form: ", ip_p`

Special Allowed IP:

1. localhost

Connection Refused

`Connection fail!`

NTLM and User Authentication

Register an already registered user

return `FAILURE("Username is already in users!")`

```
# Attempt to register a user that is already registered
if new_username in users:
    print("Username is already in users!")
    return FAILURE("Username is already in users!")
```

Success case:
`SUCCESS("Your Registered Username is " + new_username)`

Login parameter less than 2

`FAILURE(Please re-enter the login command with your username and password)`

Login parameter more than 2

`FAILURE(Password shouldn't include spaces)`

Register parameter less than 2

`FAILURE(Please re-enter the command with username and password)`

Register parameterMore than 3

`FAILURE(Username or password shouldn't include spaces)`

Login after being logged in

`FAILURE("You have logged in "+login_user+", please logout before you want to change to other user")`, login_user

```
elif msg[0] == 'login':
    print("try to login at the same time!")
    return FAILURE("You have logged in "+login_user+", please logout before you want to change to other user"), login_user
```

Register when logged in

return `FAILURE("The register command can only be done when not logging!")`, login_user

The user don't exist while log in

`FAILURE("The user does not exist!")`

The user's MD5 encrypt password doesn't match with DB log

`FAILURE("Wrong password!")`

Challenge failed

`FAILURE("Authentication Failed!")`

Register, changepwd, login with blanks

`Error on format!`
`FAILURE("Invalid password, please don't contain blank on password")`

Same Password

`FAILURE("Same password, please don't take same password!")`

Telnet Service

More number in sub and divide

`return FAILURE("Please enter Valid number! subtract $(number1) $(number2) "), login_user`
`return FAILURE("Please enter Valid number! divide $(number1) $(number2) "), login_user`

Divide 0

return `FAILURE("The dividend cannot be zero!")`, login_user

Input NAN

`FAILURE("Please enter Valid number!")`

Others

Exit with more parameters

return `FAILURE("What are you doing?")`, login_user

for example:

```
exit 1 DROP TABLE ALL
```

Disconnect

`200:disconnected`

Reference

Li, Y., Chard, R., Babuji, Y., Chard, K., Foster, I., & Li, Z. (2024). UniFaaS: Programming across distributed cyberinfrastructure with federated function serving (arXiv:2403.19257). https://arxiv.org/abs/2403.19257